

ӘЛ-ФАРАБИ АТЫНДАҒЫ ҚАЗАҚ ҰЛТТЫҚ УНИВЕРСИТЕТІ

ӘОЖ 004.415(043.3)

Қолжазба Құқығында

САРСЕМБАЕВ МАГЖАН СМАГУЛОВИЧ

**Жану процестерінің кинетикалық теңдеулерін есептеуге арналған
жоғары өнімді GPU алгоритмдерін құру
6B070400 - Есептеуіш техника және бағдарламалық жабдықтау**

Философия докторы (PhD)
ғылыми дәрежесін алу үшін дайындалған диссертация

Ғылыми кеңесшілері:

Б.А. Урмашев

ф-м.ғ.к, әл-Фараби атындағы Қазақ
Ұлттық Университеті профессоры,

Waldemar Wojcik

т.ғ.д Люблин политехникалық
университеті профессоры
(Люблин қ., Польша)

Қазақстан Республикасы
Алматы, 2026 жыл

МАЗМҰНЫ

НОРМАТИВТІ СІЛТЕМЕЛЕР	4
БЕЛГІЛЕУЛЕР МЕН ҚЫСҚАРТУЛАР.....	5
КІРІСПЕ	6
1 ЖОҒАРЫ ӨНІМДІ ТЕХНОЛОГИЯ НЕГІЗДЕРІ.....	10
1.1 Графикалық процессорларда жалпы мақсаттағы есептеулер	10
1.2 NVIDIA CUDA бағдарламалық жабдықтамасының архитектурасы	14
1.2.1 CUDA бағдарламалық-аппараттық платформасы	15
1.3 CUDA технологиясының жад моделі.....	16
1.4 Мультипроцессорлар	20
1.5 CUDA технологиясының бағдарламалау моделі	23
1.5.1 Хост процессорында CUDA бағдарламаларын ұйымдастыру	25
1.5.2 Графикалық процессорда бағдарламаны орындау моделі.....	25
1.6 Жану процестерін модельдеуде параллель есептеулерді қолдану.....	26
1.7 Серверлік есептеу технологиялары және ғылыми есептеулер	27
1.8 Жану модельдерін модельдеуге қойылатын талаптар.....	29
2 ЖОҒАРЫ ӨНІМДІ ЕСЕПТЕУ ЖҮЙЕСІН ҰЙЫМДАСТЫРУ	33
2.1 Мәселенің математикалық қойылымы.....	33
2.2 Таратылған жоғарыөнімділікті есептеу жүйелерінде қолдануға арналған кинетикалық солвер реализациясы	34
3 ЖАНУ ПРОЦЕССІН СИПАТТАЙТЫН АҚПАРАТТЫҚ-ЕСЕПТЕУ ЖҮЙЕСІН ҚҰРУ	39
3.1 Метанның жану механизмі	39
3.2 CUDA негізіндегі кинетикалық солвер алгоритмінің құрылымы.....	48
3.2.1 GPU kernel ішіндегі есептеу барысы.....	50
3.2.2 Бағдарламаның жалпы алгоритмі.....	52
3.2.3 Бағдарламаның деректер құрылымы.....	54
3.2.4 GPU жадысын ұйымдастыру	56
3.3 Runge–Kutta әдісінің параллель іске асырылуы.....	59
3.4 Параллель есептеудің артықшылықтары.....	61
3.4.1 Атомарлық операцияларды қолдану	61
3.5 Нәтижелерді сақтау.....	64
3.6 Параллель кинетикалық солверді құру сипаттамасы	70
3.7 Бағдарламалық архитектураны ұйымдастыру	81

ҚОРЫТЫНДЫ	95
ПАЙДАЛАНҒАН ӘДЕБИЕТТЕР ТІЗІМІ	98
ҚОСЫМША А.	104
ҚОСЫМША Б.....	111
ҚОСЫМША В.....	123

НОРМАТИВТІ СІЛТЕМЕЛЕР

Бұл диссертацияда келесі стандарттарға сәйкес сілтемелер қолданылды:

ҚР МЖМБС 5.04.034 – 2011 «Қазақстан Республикасының Мемлекеттік жалпыға міндетті білім беру стандарты. Жоғары оқу орнынан кейінгі білім. Докторантура». Негізгі ережелер ҚР білім және ғылым министрімен бекітілген. «17» маусым 2011 ж. №261. Астана, 2011.

«Диссертацияларды және авторефераттарды рәсімдеу бойынша нұсқаулық», ҚР БҒМ, Жоғары аттестаттау комитеті, Алматы, 2004. МЕСТ 7.1-2003. Библиографиялық жазба.

БЕЛГІЛЕУЛЕР МЕН ҚЫСҚАРТУЛАР

CUDA – Compute Unified Device Architecture, NVIDIA графикалық процессорларында параллель есептеулерді ұйымдастыру технологиясы
GPU – Graphics Processing Unit, графикалық процессор
CPU – Central Processing Unit, орталық процессор
RAM – Random Access Memory, жедел жад
GPGPU – General-Purpose computing on Graphics Processing Units, графикалық процессорларда жалпы мақсаттағы есептеулерді орындау
SM – Streaming Multiprocessor, GPU ішіндегі ағындық мультипроцессор
Thread – CUDA ағыны, GPU-дағы ең кіші орындаушы есептеу бірлігі
Block – CUDA ағындар блогы
Grid – CUDA блоктар жиыны
Global memory – GPU-дың негізгі жады аймағы
Shared memory – CUDA блогы ішіндегі жылдам ортақ жады
Register – әр CUDA ағынына тиесілі жергілікті жылдам жады
atomicAdd – бірнеше CUDA ағыны бір жады ұяшығына бір уақытта қосу операциясын орындағанда қолданылатын атомарлық қосу операциясы
Block reduction – CUDA блогы ішінде аралық нәтижелерді жинақтау әдісі
Kernel – GPU-да орындалатын CUDA функциясы
Host – CPU жағы, яғни негізгі басқарушы есептеу ортасы
Device – GPU жағы, яғни параллель есептеу орындалатын құрылғы
PCIe – Peripheral Component Interconnect Express, CPU мен GPU арасындағы деректер алмасу интерфейсі
ODE – Ordinary Differential Equation, жай дифференциалдық теңдеу
RK – Runge–Kutta method, Рунге–Кутта әдісі
RK5 – бесінші ретті Рунге–Кутта әдісі
Chemkin – химиялық кинетикалық механизмдерді сипаттауға арналған файлдық формат және бағдарламалық орта
Aramco_Mech – көмірсутекті отындардың жануын сипаттайтын химиялық кинетикалық механизм
SQL Server – деректерді сақтау және басқару үшін қолданылған деректер қоры сервері
ДҚ – деректер қоры
ДҚБЖ – деректер қорын басқару жүйесі
C# – бағдарламалау тілі, қолданбалы және басқарушы модульдерді әзірлеуде қолданылды
C++ – бағдарламалау тілі, есептеу модульдерін әзірлеуде қолданылды
ASP.NET MVC – web-қосымша құруға арналған бағдарламалық технология
Windows Forms – desktop-қосымша құруға арналған C# технологиясы
API – Application Programming Interface, бағдарламалық интерфейс
GUI – Graphical User Interface, графикалық пайдаланушы интерфейсі

КІРІСПЕ

Зерттеу тақырыбының өзектілігі

Жану процестерін дәл модельдеу энергетика, экология, өнеркәсіптік қауіпсіздік және баламалы отын түрлерін зерттеу салалары үшін маңызды ғылыми-техникалық міндеттердің бірі болып табылады. Ішкі жану қозғалтқыштары, газтурбиналық қондырғылар, жылу электр станциялары және өнеркәсіптік жану камералары сияқты жүйелердің тиімділігі мен экологиялық көрсеткіштері жану процесінің қаншалықты дәл сипатталуына тікелей байланысты [1].

Жану процесі химиялық реакциялар, жылу және масса алмасу, газ динамикасы, көпкомпонентті қоспалардың өзара әрекеттесуі және реакция өнімдерінің түзілуі сияқты күрделі физика-химиялық құбылыстармен сипатталады. Мұндай процестерді толық сипаттау үшін химиялық кинетиканың дифференциалдық теңдеулер жүйесін шешу қажет. Алайда нақты жану механизмдері көптеген химиялық компоненттерден және үлкен реакциялар жиынтығынан тұрады. Осыған байланысты кинетикалық теңдеулерді есептеу үлкен есептеу ресурстарын және жоғары өнімді сандық алгоритмдерді талап етеді [2].

Дәстүрлі CPU негізіндегі есептеу әдістері күрделі химиялық кинетикалық жүйелер үшін әрқашан жеткілікті тиімді бола бермейді. Реакциялар саны артқан сайын есептеу уақыты айтарлықтай өседі, бұл үлкен механизмдерді қолдана отырып жану процестерін практикалық тұрғыдан қолайлы уақытта модельдеуді қиындатады. Әсіресе көпкомпонентті көмірсутекті отындардың жануын, зиянды шығарындылардың түзілуін, азот оксидтері мен күйе бөлшектерінің пайда болуын модельдеу кезінде есептеу жүктемесі едәуір артады [3].

Осы мәселелерді шешудің тиімді бағыттарының бірі — химиялық кинетикалық механизмдерді оңтайландыру және есептеу шығынын азайтатын арнайы бағдарламалық құралдарды қолдану. Мұндай тәсілдер күрделі механизмдерді қысқартуға, реакциялар жүйесін тиімді өңдеуге және сандық модельдеуді жылдамдатуға мүмкіндік береді [4, 5].

Сонымен қатар жоғары өнімді параллель есептеу технологияларын қолдану химиялық кинетика теңдеулерін шешудің маңызды бағыты болып табылады. GPU архитектурасы бір уақытта көптеген есептеу ағындарын орындауға мүмкіндік береді, ал химиялық кинетика есептерінде реакциялар бойынша орындалатын көптеген операциялар бір-бірінен тәуелсіз болуы мүмкін. Сондықтан мұндай есептерді GPU ағындары арасында параллель үлестіру есептеу уақытын қысқартудың тиімді жолы болып табылады [6].

Жану процестерін жоғары өнімді есептеу жүйелерінде модельдеу үшін реакциялық ағындарды сипаттайтын заманауи есептеу кешендері мен адаптивті тор әдістері де кеңінен қолданылады. Мұндай тәсілдер күрделі жану құбылыстарын кеңістіктік және уақыттық тұрғыдан неғұрлым дәл сипаттауға мүмкіндік береді [7, 8].

Осылайша, жану процестерінің кинетикалық теңдеулерін есептеуге арналған жоғары өнімді GPU алгоритмдерін құру қазіргі ғылыми және

қолданбалы есептер үшін өзекті бағыт болып табылады.

Жұмыстың мақсаты: жану процестерінің кинетикалық теңдеулерін шешуге арналған жоғары өнімді GPU алгоритмдерін әзірлеу, оларды бағдарламалық жүзеге асыру және есептеу тиімділігін арттыру.

Зерттеудің міндеттері.

- Жану процестерінің кинетикалық модельдерін талдау, сандық шешу әдістерін зерттеу

- GPU архитектурасына бейімделген параллель алгоритмдерді құру

- CUDA негізінде бағдарламалық жүзеге асыру, жадыны пайдалану мен есептеу үдерісін оңтайландыру, алгоритмдердің дәлдігі мен өнімділігін бағалау

- CPU және GPU тәсілдерінің салыстырмалы талдауын жүргізу

Зерттеу әдістері

Зерттеу барысында химиялық кинетика теориясының әдістері, математикалық модельдеу, сандық интегралдау әдістері, Рунге–Кутта әдісі, параллель есептеу әдістері, CUDA бағдарламалау технологиясы, GPU жадысын оңтайландыру тәсілдері, CPU және GPU есептеу уақыттарын салыстырмалы талдау, сондай-ақ есептеу тәжірибелерінің нәтижелерін бағалау әдістері қолданылды.

Зерттеу объектісі

Жану процестерін сипаттайтын химиялық кинетикалық механизмдер және олардың негізінде қалыптасатын кинетикалық теңдеулер жүйесі.

Зерттеу пәні

Жану процестерінің кинетикалық теңдеулерін GPU архитектурасында параллель есептеу алгоритмдері, CUDA негізіндегі кинетикалық солверді бағдарламалық іске асыру әдістері және GPU жадысын тиімді ұйымдастыру тәсілдері.

Алынған нәтижелердің ғылыми жаңалығы

Алғаш рет жану процестерін модельдеу есептерінде 162 компоненттен және 1006 реакциядан тұратын химиялық кинетикалық механизмнің теңдеулер жүйесін GPU архитектурасында шешуге арналған жоғары өнімді алгоритм әзірленіп, CUDA негізіндегі алғашқы есептеу кезеңінде CPU әдісімен салыстырғанда 17,5 есе жылдамдату алынды, ал кейін GPU жадысын тиімді ұйымдастыру, shared memory қолдану және есептеу үдерісін оңтайландыру нәтижесінде 493 компоненттен және 2716 реакциядан тұратын күрделі механизм үшін есептеу жылдамдығы CPU әдісімен салыстырғанда 154,89 есеге дейін артты.

Қорғауға шығарылатын тұжырымдар

Ұсынылған GPU алгоритмдері CPU негізіндегі әдістермен салыстырғанда алдымен 17,5 есе жылдамдату көрсетсе, GPU жадысын shared memory арқылы оңтайландыру нәтижесінде есептеу жылдамдығы 154,89 есеге дейін артып, жоғары өнімділікке қол жеткізілді.

Зерттеудің теориялық және практикалық маңыздылығы

Теориялық маңыздылығы: зерттеу нәтижелері химиялық кинетика теңдеулерін жоғары өнімді GPU архитектурасында параллель есептеу

әдістерінің дамуына үлес қосады. Жұмыста реакциялар жүйесін CUDA ағындары арасында үлестіру, GPU жады иерархиясын пайдалану және Рунге–Кутта әдісін параллель іске асырудың теориялық және алгоритмдік негіздері қарастырылды.

Практикалық маңыздылығы: әзірленген алгоритмдер мен бағдарламалық шешімдер күрделі жану механизмдерін қысқа уақытта есептеуге мүмкіндік береді. Бұл тәсіл жану процестерін модельдеу, баламалы отындарды зерттеу, энергетикалық қондырғыларды оңтайландыру, зиянды шығарындыларды бағалау және жоғары өнімді ғылыми есептеулер салаларында қолданылуы мүмкін.

Диссертациялық жұмыстың көлемі мен құрылымы. Диссертация кіріспеден, 3 бөлімнен, қорытындыдан, пайдаланылған әдебиеттер тізімі және қосымшадан тұрады. Диссертациялық жұмыстың жалпы көлемі 123 бет, 43 сурет, 3 кесте, 47 формула бар. Әдебиеттер тізімі 74 тақырыптан тұрады.

Жұмыстың апробациясы

Жұмыстың нәтижелері әл-Фараби атындағы Қазақ ұлттық университетінің компьютерлік ғылымдар кафедрасы мен ақпараттық технологиялар факультетінің ғылыми семинарларында талқыланып, сонымен қатар келесі конференцияларда баяндалған:

- Сарсембаев М.С., Урмашев Б.А. Разработка кинетического механизма горения метана с образованием оксидов азота и полиароматических молекул // Материалы Международной конференции «Инновационное развитие и востребованность науки в современном Казахстане». – Алматы, 2017.

- Сарсембаев М.С., Макашев Е.П., Урмашев Б.А., Исахов А.А. Влияние сверхзвукового потока температуры на диффузионное горение водорода // Материалы III Международной конференции «Промышленные технологии и инженерия». – Шымкент, 2016.

- Сарсембаев М.С., Урмашев Б.А. Создание кинетического механизма горения метана с образованием оксидов азота // Материалы II Международной научно-практической конференции «Modern Methodology of Science and Education». – Варшава, Польша, 2018.

Жарияланымдар. Диссертация тақырыбы бойынша 4 баспа жұмысы жарияланды, оның ішінде 3 жұмыс ҒЖБССҚК ҚР ұсынған басылымдарда, 1 жұмыс халықаралық дәйексөздер базасына кіретін рецензияланатын журналдарда «Web of Science» және/немесе «SCOPUS», 3 жұмыс халықаралық конференциялар жинақтарында жарияланды. Сонымен қатар, диссертациялық зерттеу нәтижелері негізінде келесі авторлық құқық объектісі тіркелді: «Информационно-вычислительная система для автоматизированной генерации химических моделей описывающих горение реальных топлив» атты авторлық құқық объектісі

Қазақстан Республикасының авторлық құқықпен қорғалатын объектілерге құқықтардың мемлекеттік тізіліміне мәліметтерді енгізу туралы 2020 жылғы 17 қараша берілген № 13255 куәлік тіркелген (Қосымша В).

Зерттеудің ғылыми жобалармен байланысы. Диссертациялық жұмыс

келесі ғылыми жобалар аясында орындалды:

– «Нақты отынның жануын сипаттайтын химиялық модельдерді автоматты генерациялауға арналған бұлттық ақпараттық-есептеу жүйесінің теориялық негіздерін және прототипін әзірлеу» (**ҒЗЖ №0115РК00770, 2015–2017 жж.**);

– «Фармакокинетика мен фармакодинамика үлгілерін автоматты әзірлеу және оңтайландыру үшін бұлттық ақпараттық-есептеу кешенін жасау және енгізу» (**IRN жобасы: AP051329, 2018–2020 жж.**).

1 ЖОҒАРЫ ӨНІМДІ ТЕХНОЛОГИЯ НЕГІЗДЕРІ

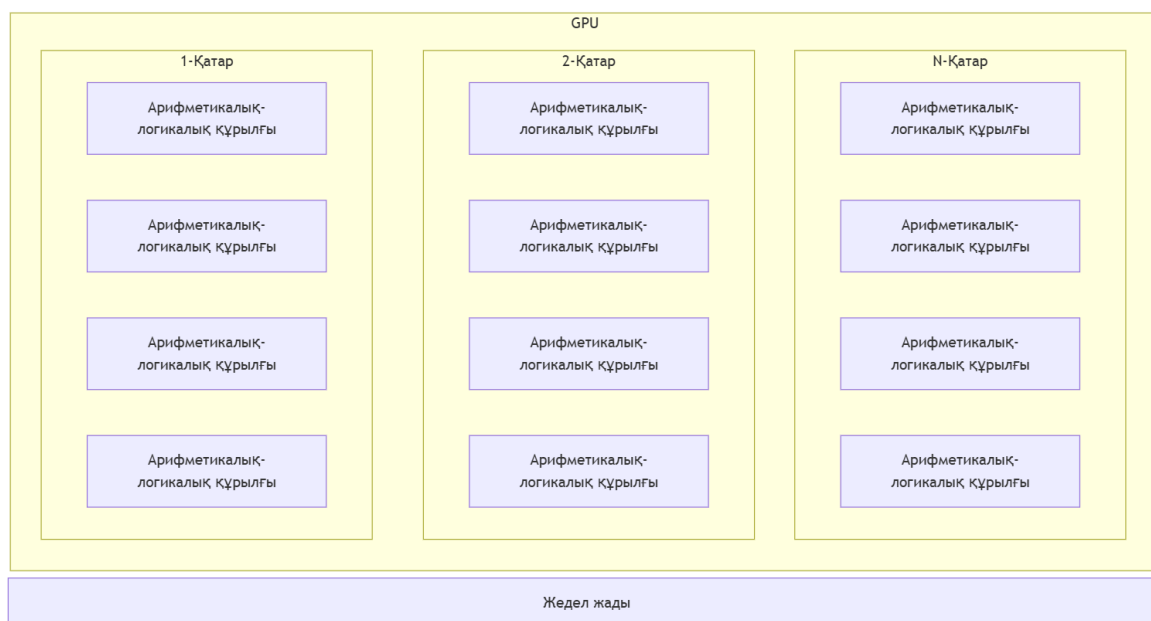
1.1 Графикалық процессорларда жалпы мақсаттағы есептеулер

GPGPU («General-Purpose Graphics Processing Unit» – жалпы мақсаттағы графикалық процессорларды пайдалану технологиясы) – әдетте орталық процессор орындайтын жалпы есептеулерді жеделдету үшін графикалық процессорларды қолдануға мүмкіндік беретін аппараттық және бағдарламалық технологиялар жиынтығы болып табылады [9].

Graphics Processing Units (GPU) – күрделі есептеулерді орындауға және деректердің өте жоғары өткізу қабілетін қамтамасыз етуге арналған жоғары өнімді көп ядролы процессорлар. GPGPU технологиясының дамуы шейдерлік бағдарламалардың және Cg, GLSL, HLSL сияқты жоғары деңгейлі бағдарламалау тілдерінің пайда болуымен байланысты.

Графикалық процессорлардың бастапқы мақсаты графикалық деректерді өңдеуге бағытталған арнайы есептерді орындау болды. Осыған байланысты орталық процессор мен графикалық процессордың архитектуралары айтарлықтай ерекшеленеді (CPU және GPU құрылымдары 1, 2 - суретте көрсетілген).

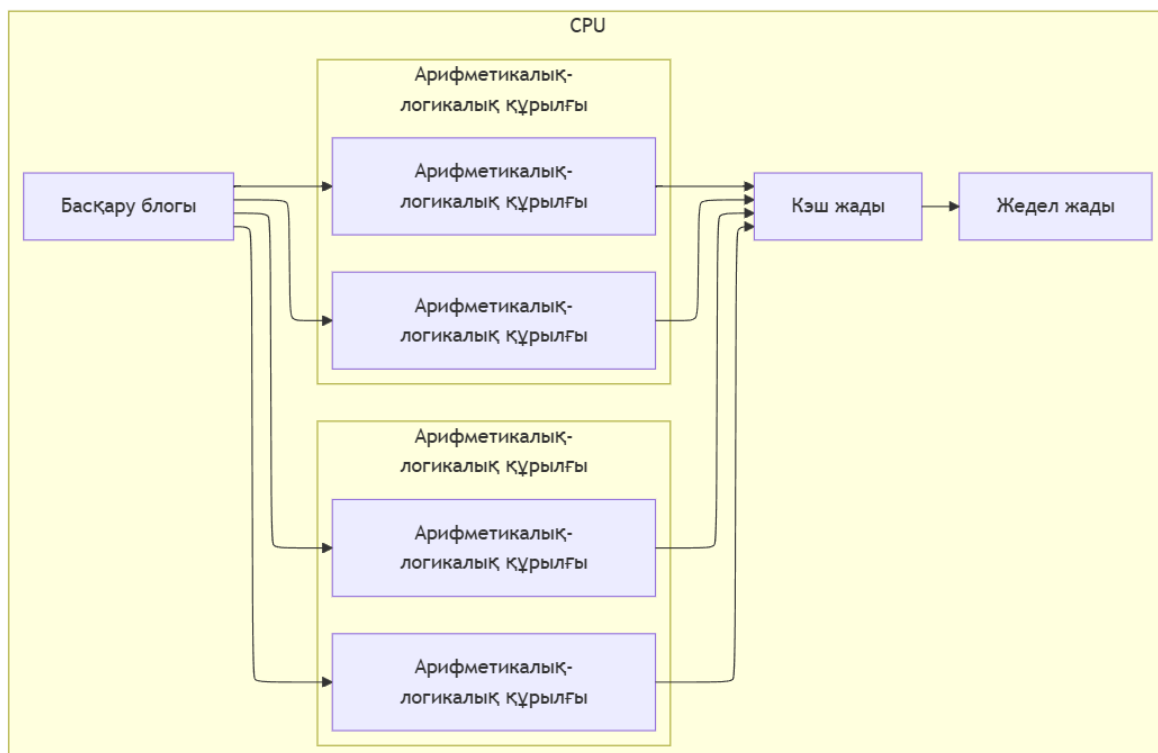
Мысалы, NVIDIA компаниясының графикалық процессорлары көптеген есептеу ядроларынан тұратын мультипроцессорлық архитектураға негізделген және олардың әрқайсысы мыңдаған регистрлермен жұмыс істей алады.



Сурет 1 – GPU құрылымындағы арифметикалық-логикалық блоктардың ұйымдастырылуы

GPU SIMD (Single Instruction, Multiple Data – бір нұсқаулық ағыны және бірнеше деректер ағыны) есептеу әдісін қолдайды. Бұл әдіс бойынша графикалық процессордың көптеген есептеу ядролары әртүрлі деректер жиындары үшін бірдей нұсқауларды орындайды. Көптеген графикалық алгоритмдер осы тәсілді графикалық есептерді тиімді орындаудың негізгі құралы ретінде пайдаланады. Кіріс деректер ағынын өңдеп, оны шығыс деректер

ағынына түрлендіретін мұндай есептеу модулі ядро (kernel) деп аталады [10].



Сурет 2 – GPU және CPU құрылғысы. Сол жақта - процессор, оң жақтағы графикалық процессор

Орталық және графикалық процессорлардың архитектуралары арасындағы негізгі айырмашылықтарды қорытындылай отырып, орталық процессордың басты міндеті – бір нұсқаулық ағынын дәйекті түрде мүмкіндігінше жоғары өнімділікпен орындау екенін атап өтуге болады. Ал GPU архитектурасы бір уақытта көптеген параллель ағындарды орындауға бағытталған.

Орталық процессордың өнімділігін арттыру үшін әзірлеушілер бір такт ішінде орындалатын операциялар санын көбейтуге ұмтылады. Алайда CPU орындайтын нұсқаулықтар ағыны көбіне дәйекті сипатқа ие. Сондықтан қазіргі архитектураларда өнімділікті арттырудың негізгі тәсілдерінің бірі – есептеу ядроларының санын көбейту және параллель орындауды кеңейту болып табылады.

Графикалық есептеулерде деректердің көп бөлігі бір-біріне тәуелсіз өңделеді. Сондықтан графикалық конвейердің әр кезеңінде мәліметтер параллель түрде өңделе алады. Бұл графикалық процессорлардың негізгі ерекшеліктерінің бірі болып табылады. Орталық процессорда нұсқаулықтар ағыны көбінесе дәйекті түрде орындалса, графикалық процессор көптеген біртекті операцияларды бір уақытта орындауға арналған жеңілдетілген есептеу блоктарын қолданады.

Орталық және графикалық процессорларда жадыға қол жеткізуді ұйымдастыру принциптері де айтарлықтай ерекшеленеді. CPU

архитектурасында жадыға қол жеткізудің кешігуін азайту үшін кэш жадысы және тармақталуды болжау (branch prediction) сияқты технологиялар кеңінен қолданылады. Ал GPU архитектурасында жадқа қол жеткізудің кешігуі басқа тәсілмен азайтылады. Егер белгілі бір есептеу ағыны жадыдан деректерді күтіп тұрса, графикалық процессор бірден басқа дайын ағынды орындауға ауысады. Бұл тәсіл есептеу құрылғыларының бос тұрмауына мүмкіндік береді.

Сонымен қатар графикалық процессорлардың жады өткізу қабілеті орталық процессорлармен салыстырғанда айтарлықтай жоғары болады. GPU архитектурасында кэш жадысы да қолданылады, бірақ оның көлемі әдетте салыстырмалы түрде шағын болады (шамамен 128–256 КБ). Бұл өз кезегінде жадының өткізу қабілеттілігін арттыруға мүмкіндік береді.

Графикалық процессорларда көпағындылық (multithreading) аппараттық деңгейде жүзеге асырылады. Орталық процессорда ағындар арасында ауысу белгілі бір уақыт шығындарын талап етеді, себебі контексті ауыстыру бірнеше такт уақытын алады. Сонымен қатар CPU ядросы әдетте бір уақытта шектеулі ғана ағындарды орындай алады [11–12]. Ал GPU архитектурасында ағындар арасында ауысу өте жылдам жүзеге асырылады және әрбір есептеу блогы жүздеген, кей жағдайда мыңға жуық ағындарды бір уақытта өңдей алады.

Орталық процессор жалпы мақсаттағы есептеу құрылғысы ретінде әртүрлі типтегі есептерді тиімді орындай алады. Ал графикалық процессорлар негізінен параллель есептеулерге бағытталған арнайы архитектураға ие. Сондықтан тармақталуы көп және басқару логикасы күрделі есептерде GPU тиімділігі CPU-мен салыстырғанда төмен болуы мүмкін.

Осыған байланысты GPGPU технологиясы әсіресе үлкен көлемдегі деректерді өңдеу кезінде жоғары тиімділік көрсетеді. GPGPU технологиясының алғашқы даму кезеңінде GPU көмегімен орындалатын есептеулердің жылдамдығы кейбір жағдайларда орталық процессорлармен салыстырғанда шамамен он есеге дейін жоғары болған [13].

Сондықтан графикалық процессорларды өндіруші компаниялар GPGPU технологиясының болашағына үлкен үміт артып, осы бағыттағы зерттеулер мен әзірлемелерді қарқынды түрде дамытуда.

Қазіргі уақытта графикалық үдеткіштердің есептеу өнімділігі айтарлықтай артты және олар жоғары өнімді есептеу жүйелерінің негізгі компоненттерінің біріне айналды. Соңғы буындағы графикалық процессорлар секундына триллиондаған операцияларды орындауға қабілетті. Мысалы, NVIDIA компаниясының 2022 жылы ұсынылған NVIDIA H100 графикалық үдеткіші қос дәлдіктегі есептеулер үшін шамамен 60 TFLOPS өнімділікке ие. Ал тұтынушылық сегментке арналған NVIDIA GeForce RTX 4090 графикалық картасының бір дәлдіктегі есептеулердегі өнімділігі 80 TFLOPS-тен жоғары деңгейге жетеді [14].

Салыстыру үшін қазіргі заманғы орталық процессорлардың есептеу өнімділігі айтарлықтай төмен болады. Мысалы, серверлік сегментке арналған Intel Xeon немесе AMD EPYC процессорларының есептеу өнімділігі әдетте бірнеше TFLOPS деңгейінде ғана болады. Сондықтан көптеген ғылыми,

инженерлік және модельдеу есептерінде параллель есептеулерді орындау үшін графикалық процессорларды қолдану тиімді болып табылады.

Графикалық процессорлардың жоғары өнімділігі олардың архитектуралық ерекшеліктерімен түсіндіріледі. GPU архитектурасы жүздеген немесе мыңдаған жеңіл есептеу ядроларынан тұрады және олар бір уақытта көптеген параллель есептеу ағындарын орындай алады. Бұл архитектура үлкен көлемдегі деректерді бір уақытта өңдеуді қажет ететін есептер үшін өте тиімді. Сондықтан қазіргі таңда GPU құрылғылары жасанды интеллект, машиналық оқыту, физикалық модельдеу, биоинформатика, климаттық модельдеу және жоғары өнімді есептеулер салаларында кеңінен қолданылады.

Соңғы жылдары графикалық процессорлар жоғары өнімді есептеу жүйелерінде маңызды рөл атқара бастады. Мысалы, әлемдегі ең қуатты суперкомпьютерлердің бірі – Frontier суперкомпьютері (АҚШ, Oak Ridge ұлттық зертханасы) AMD GPU үдеткіштерін пайдалана отырып 1 эксафлопстан (10^{18} операция/с) асатын өнімділікке қол жеткізді. Бұл көрсеткіш графикалық процессорлардың жоғары өнімді есептеулердегі маңызын айқын көрсетеді.

Графикалық процессорларды жалпы мақсаттағы есептеулерде қолдану GPGPU (General-Purpose computing on Graphics Processing Units) технологиясының дамуына алып келді. Бұл технология графикалық процессорларды тек графиканы өңдеу үшін ғана емес, сонымен қатар ғылыми және инженерлік есептерді шешу үшін пайдалануға мүмкіндік береді. GPGPU технологиясы үлкен көлемдегі деректерді параллель өңдеу қажет болатын есептерде есептеу уақытын айтарлықтай қысқартуға мүмкіндік береді.

Графикалық процессорларды жалпы мақсаттағы есептеулерде қолдану бағыты 2000-жылдардың басында белсенді түрде дами бастады. 2003 жылдан бастап графикалық процессор өндірушілері графикалық емес есептеулерді қолдайтын арнайы бағдарламалық платформаларды дамытуға кірісті. Осы бағыттағы ең маңызды шешімдердің бірі – NVIDIA компаниясы әзірлеген CUDA (Compute Unified Device Architecture) платформасы болып табылады.

CUDA платформасы GPU архитектурасының есептеу мүмкіндіктерін жалпы мақсаттағы есептеулер үшін пайдалануға мүмкіндік береді. CUDA көмегімен бағдарламашылар C, C++ немесе Python сияқты бағдарламалау тілдерін пайдалана отырып графикалық процессорларда параллель есептеулерді жүзеге асыра алады. Бұл технология ғылыми модельдеу, деректерді өңдеу, жасанды интеллект және жоғары өнімді есептеу жүйелерінде кеңінен қолданылады[15].

Сонымен қатар GPU негізіндегі есептеулерді жүзеге асыру үшін OpenCL (Open Computing Language) сияқты ашық стандарттар да қолданылады. OpenCL әртүрлі аппараттық платформаларда, соның ішінде GPU, CPU және басқа да есептеу құрылғыларында параллель есептеулерді орындауға мүмкіндік береді. Алайда қазіргі уақытта CUDA технологиясы өзінің жоғары өнімділігі мен дамыған бағдарламалық экожүйесінің арқасында ғылыми және өндірістік есептеулерде кеңінен қолданылады.

Осылайша графикалық процессорларды жалпы мақсаттағы есептеулерде

қолдану қазіргі заманғы жоғары өнімді есептеу жүйелерінің дамуына үлкен үлес қосып отыр. GPU архитектурасының параллель есептеулерге бейімділігі үлкен көлемдегі деректерді өңдеуді қажет ететін есептерді тиімді шешуге мүмкіндік береді.

1.2 NVIDIA CUDA бағдарламалық жабдықтамасының архитектурасы

CUDA (Compute Unified Device Architecture) – NVIDIA компаниясы әзірлеген бағдарламалық-аппараттық платформа. Ол GPGPU технологиясын қолдайтын графикалық процессорлардың есептеу мүмкіндіктерін жалпы мақсаттағы есептерді орындау үшін пайдалануға мүмкіндік береді [16]. Бұл технология графикалық процессорлардың ресурстарын тек графиканы өңдеу үшін ғана емес, сонымен қатар ғылыми және инженерлік есептеулерде қолдануға жол ашады.

Графикалық процессорлар көптеген арифметикалық-логикалық құрылғылардан тұрады және олар бір уақытта үлкен көлемдегі параллель есептеулерді орындауға қабілетті. Мұндай ұйымдастыру деректердің үлкен массивтерін өңдеуді қажет ететін есептер үшін өте тиімді болып табылады.

CUDA технологиясының негізгі мақсаты – графикалық процессорларды жалпы мақсаттағы есептеулерде қолдану мүмкіндігін қамтамасыз ету. Осыған байланысты GPU есептеу блоктарына жадыға кездейсоқ қол жеткізу, сондай-ақ бағдарламалық түрде басқарылатын кэш-жадты пайдалану мүмкіндігі берілген.

CUDA бағдарламалау ортасының маңызды ерекшеліктерінің бірі – бағдарламалау процесінің қарапайымдылығы. NVIDIA компаниясы стандартты C/C++ бағдарламалау тілін негізге алып, оны параллель есептеулерді ұйымдастыруға арналған арнайы кеңейтулермен толықтырды. Нәтижесінде бағдарламашылар графикалық процессорларда орындалатын есептеу алгоритмдерін тиімді түрде іске асыра алады.

CUDA бағдарламалау моделінде есептеу ағындары иерархиялық түрде ұйымдастырылады. Ең төменгі деңгейде жеке ағындар (threads) орналасады. Бұл ағындар блоктарға (blocks), ал блоктар торларға (grids) біріктіріледі. Осындай құрылым параллель есептеулерді тиімді басқаруға және GPU ресурстарын толық пайдалануға мүмкіндік береді.

CUDA технологиясын қолдану үшін арнайы компилятор мен драйверлер қолданылады. Олар графикалық процессордың есептеу мүмкіндіктерін тікелей пайдалануға мүмкіндік береді. Осылайша бағдарламашыларға OpenGL немесе DirectX сияқты графикалық интерфейстерді қолданудың қажеті болмайды [17].

GPU негізіндегі параллель есептеулерді жүзеге асыру үшін басқа да технологиялар бар. Соның бірі – AMD компаниясының AMD STREAM технологиясы. Бұл технология OpenCL (Open Computing Language) стандартына негізделген және әртүрлі аппараттық платформаларда параллель есептеулерді орындауға мүмкіндік береді.

AMD STREAM технологиясы ECC (Error Correcting Code) жадысын қолдауды, қалқымалы нүктелік операцияларды жоғары жылдамдықпен орындауды және бірнеше GPU арасында тиімді деректер алмасуды қамтамасыз етеді. Сонымен қатар GPU-ға оңтайландырылған кітапханалар мен үшінші тарап

бағдарламалық құралдарымен біріктірілуі AMD FirePro графикалық карталарын жоғары өнімді есептеулерде қолдануға мүмкіндік береді [18].

Осылайша CUDA платформасы графикалық процессорлардың параллель есептеу мүмкіндіктерін тиімді пайдалануға арналған маңызды құрал болып табылады. Қазіргі уақытта бұл технология ғылыми модельдеу, деректерді талдау, жасанды интеллект, машиналық оқыту және жоғары өнімді есептеу жүйелерінде кеңінен қолданылады.

1.2.1 CUDA бағдарламалық-аппараттық платформасы

CUDA (Compute Unified Device Architecture) – NVIDIA компаниясы әзірлеген бағдарламалық-аппараттық платформа болып табылады. Бұл платформа графикалық процессорлардың есептеу мүмкіндіктерін жалпы мақсаттағы есептеулерде пайдалануға мүмкіндік береді және GPGPU (General-Purpose Computing on Graphics Processing Units) технологиясын іске асыруға арналған. CUDA технологиясының алғашқы нұсқасы 2007 жылдың 15 ақпанында ұсынылды.

CUDA платформасының негізгі мақсаты – бағдарламашыларға графикалық процессорларды параллель есептеулерді орындауға арналған тиімді есептеу құрылғысы ретінде қолдануға мүмкіндік беру. Осы технологияның көмегімен GPU тек графикалық есептерді орындау үшін ғана емес, сонымен қатар ғылыми, инженерлік және деректерді өңдеу есептерін шешу үшін де қолданылады. Сонымен қатар CUDA бағдарламалау ортасы бағдарламашыларды графикалық API-лармен (OpenGL немесе DirectX) жұмыс істеу қажеттілігінен босатады.

Графикалық процессорлар бастапқыда графикалық деректерді өңдеуге арналған құрылғылар ретінде дамығандықтан, олардың архитектурасы орталық процессорлардан айтарлықтай ерекшеленеді. Сондықтан CUDA платформасымен алғаш жұмыс істеген кезде GPU архитектурасының ерекшеліктерін түсіну белгілі бір қиындықтар тудыруы мүмкін.

CUDA платформасының құрамына бірнеше негізгі компоненттер кіреді. Олардың қатарына графикалық процессор, CUDA Toolkit, CUDA драйвері және қосымша бағдарламалық кітапханалар жатады. CUDA Toolkit құрамында платформамен жұмыс істеуге қажетті бағдарламалық құралдар жиынтығы бар. Оған параллель бағдарламаларды құрастыруға арналған nvcc компиляторы, сондай-ақ GPU есептеу мүмкіндіктерін пайдалануға арналған кітапханалар кіреді [19].

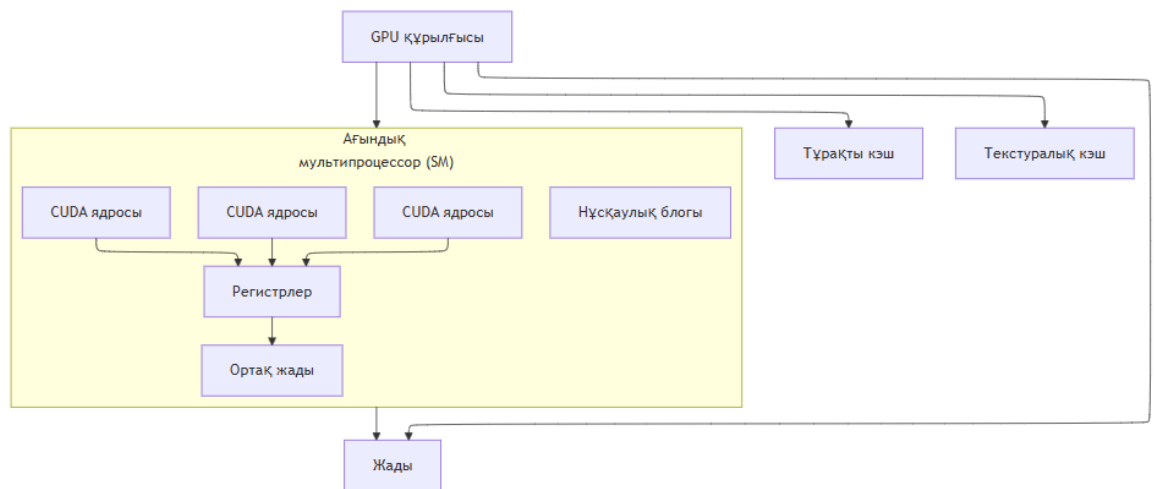
nvcc компиляторы CUDA тілінде жазылған бағдарламалардың бастапқы кодын графикалық процессорда орындалатын машиналық кодқа түрлендіруге арналған. Сонымен қатар CUDA Toolkit құрамында параллель есептеулерді жүзеге асыруға арналған қосымша бағдарламалық кітапханалар мен құралдар бар.

CUDA драйвері графикалық процессор мен бағдарламалық қамтамасыз ету арасындағы өзара әрекеттесуді қамтамасыз етеді. Бұрынғы нұсқаларында CUDA драйвері аралық кодты GPU-да орындалатын микрокодқа түрлендіруге жауап беретін компоненттерді қамтыған. Қазіргі уақытта CUDA драйвері графикалық

құрылғы драйверімен бірге жұмыс істейтін бірыңғай жүйе ретінде ұйымдастырылған.

CUDA платформасын меңгеруді жеңілдету үшін NVIDIA компаниясы CUDA SDK құрамына мысал бағдарламалар жиынтығын енгізген. Бұл мысалдар CUDA архитектурасының негізгі мүмкіндіктерін көрсетуге және параллель бағдарламалаудың негізгі қағидаларын түсінуге көмектеседі.

CUDA технологиясын NVIDIA компаниясының көптеген графикалық процессорлары қолдайды. Олардың қатарына GeForce, Quadro және Tesla серияларындағы құрылғылар жатады. Бұл платформаның негізгі ерекшеліктеріне графикалық API-лармен (OpenGL және DirectX) өзара әрекеттесу мүмкіндігі, жадқа тікелей қол жеткізу, 32 және 64 биттік операциялық жүйелерді қолдау, сондай-ақ C/C++ тілінің кеңейтілген нұсқасын пайдалану жатады.



Сурет 3 – CUDA архитектурасындағы графикалық процессордың құрылымы

3 суретте CUDA платформасының жалпы архитектурасы көрсетілген. Платформаның жұмыс принципін толық түсіну үшін оның логикалық архитектурасын қарастыру қажет [20].

1.3 CUDA технологиясының жад моделі

CUDA технологиясында жадқа қол жеткізу механизмі параллель есептеулердің тиімділігін анықтайтын негізгі факторлардың бірі болып табылады. GPU архитектурасында байттық адрестеу мүмкіндігі бар бірнеше жад кеңістігі қолданылады. Бұл жад түрлері әртүрлі есептеу сценарийлеріне оңтайландырылған және CUDA ағындарының деректермен тиімді жұмыс істеуіне мүмкіндік береді.

Әрбір ағынның өзіне тиесілі **жергілікті жады (local memory)** болады. Бұл жады тек сол ағынға ғана қолжетімді және басқа ағындар үшін көрінбейді. Жергілікті жады әдетте ағынның жеке айнымалыларын сақтау үшін қолданылады.

Әрбір ағын блогы **ортақ жадыны (shared memory)** пайдалана алады. Ортақ жады блок ішіндегі барлық ағындарға ортақ болып табылады және оның

өмір сүру уақыты ағын блогының орындалу уақытымен шектеледі. Бұл жад түрі ағындар арасында деректер алмасуды ұйымдастыру үшін қолданылады және жадыға қол жеткізу жылдамдығы өте жоғары болады.

CUDA құрылғысындағы барлық ағындар **ғаламдық жадыға (global memory)** қол жеткізе алады. Ғаламдық жады GPU құрылғысының негізгі жады болып табылады және үлкен көлемдегі деректерді сақтау үшін қолданылады. Дегенмен, бұл жад түріне қол жеткізу уақыты басқа жад түрлерімен салыстырғанда баяу болады.

Сонымен қатар CUDA архитектурасында тек оқу режимінде қолданылатын екі арнайы жад кеңістігі бар: **тұрақты жады (constant memory)** және **текстуралық жады (texture memory)**. Тұрақты жады барлық ағындар үшін ортақ болып табылады және өзгермейтін деректерді сақтау үшін пайдаланылады. Ал текстуралық жады арнайы кэштеу механизмдерін қолдану арқылы деректерге тиімді қол жеткізуді қамтамасыз етеді.

Ғаламдық, тұрақты және текстуралық жад түрлері әртүрлі қолдану сценарийлеріне оңтайландырылған. Мысалы, текстуралық жады әртүрлі адрестеу режимдерін қолдайды және кейбір деректер форматтары үшін аппараттық деңгейде сүзгілеу (filtering) операцияларын орындауға мүмкіндік береді.

CUDA архитектурасында жадқа қол жеткізудің тиімділігі адрестердің орналасуына байланысты болады. Егер жады адрестеріне қол жеткізу реттелген түрде орындалса, онда жадының өткізу қабілеттілігі жоғары болады. Ал егер адрестер ретсіз немесе шашыраңқы орналасқан болса, онда жадқа қол жеткізу тиімділігі төмендейді. Сондықтан CUDA бағдарламаларын әзірлеу кезінде жадқа қол жеткізуді оңтайландыру есептеу өнімділігін арттырудың маңызды факторларының бірі болып табылады [21].

Global жадысы

CUDA архитектурасында ғаламдық жад (global memory) графикалық процессордың негізгі құрылғы жадында орналасады және барлық ағындар үшін қолжетімді болып табылады. Бұл жад үлкен көлемдегі деректерді сақтау үшін пайдаланылады, бірақ оған қол жеткізу уақыты басқа жад түрлерімен салыстырғанда баяу болады.

Ғаламдық жадқа қол жеткізу арнайы жад транзакциялары (memory transactions) арқылы жүзеге асырылады. Бұл транзакциялар әдетте 32, 64 немесе 128 байттық сегменттер бойынша орындалады. Транзакциялар тиімді орындалуы үшін жад адрестері табиғи түрде тураланған болуы қажет. Яғни жад сегментінің бастапқы адресі оның өлшеміне еселік болуы тиіс.

CUDA ядросы ғаламдық жадқа қол жеткізетін нұсқаулықты орындаған кезде бірнеше ағынның жадқа сұраныстары бір немесе бірнеше жад транзакцияларына біріктіріледі. Бұл процесс жадқа біріктірілген қол жеткізу (coalesced memory access) деп аталады. Транзакциялардың саны әрбір ағын пайдаланатын деректердің өлшеміне және жад адрестерінің орналасуына байланысты болады.

Жалпы жағдайда транзакциялар саны неғұрлым көп болса, жадқа қол

тұрғыдан ағынға тиесілі болғанымен, физикалық түрде құрылғының ғаламдық жадында орналасады. Сондықтан жергілікті жадқа қол жеткізу жылдамдығы ғаламдық жадқа қол жеткізу жылдамдығымен шамалас.

Компилятор кейбір айнымалыларды автоматты түрде жергілікті жадқа орналастыруы мүмкін. Мұндай жағдайлар келесі жағдайларда пайда болады:

- тұрақты индекстер арқылы анықталмайтын массивтер қолданылған кезде;
- регистрлерде сақтауға тым үлкен құрылымдар немесе массивтер пайдаланылған кезде;
- ядро бағдарламасы қолжетімді регистрлер санынан артық регистр қолданған жағдайда.

Айнымалылардың жергілікті жадта орналасқанын анықтау үшін PTX ассемблер кодын талдауға болады. PTX кодында мұндай айнымалылар `.local` мнемоникасы арқылы жарияланады және оларға `ld.local` және `st.local` нұсқаулықтары арқылы қол жеткізіледі. Сонымен қатар компиляция кезінде `--ptxas-options=-v` параметрін пайдалану арқылы әрбір CUDA ядросы үшін қолданылатын жергілікті жад көлемі туралы ақпарат алуға болады.

Жергілікті жады құрылғы жадында орналасқандықтан, оған қол жеткізу кезінде ғаламдық жадтағы сияқты жад транзакциялары орындалады. Сондықтан жадқа қол жеткізудің тиімділігі адрестердің орналасуына және ағындардың жадқа қол жеткізу үлгісіне байланысты болады.

CUDA архитектурасында жергілікті жадқа қол жеткізу белгілі бір жағдайда тиімді біріктірілуі мүмкін. Егер бірнеше ағын бірдей салыстырмалы адрестерге қол жеткізсе (мысалы, массивтің бірдей индексіне немесе құрылымның бірдей өрісіне), онда жадқа қол жеткізу біріктірілген (coalesced) түрде орындалады.

Әртүрлі GPU архитектураларында жергілікті жадқа қол жеткізу кэш механизмдері арқылы оңтайландырылады. Мысалы, есептеу мүмкіндігі 2.x және 3.x архитектураларында жергілікті жадқа қол жеткізу L1 және L2 кэштері арқылы жүзеге асырылады. Ал 5.x және 6.x архитектураларында жергілікті жадқа қол жеткізу негізінен L2 кэш арқылы кэштеледі.

CUDA архитектурасында жоғары өнімділікке қол жеткізу үшін жад иерархиясының басқа элементтері де маңызды рөл атқарады. Мысалы, ортақ жады (shared memory) графикалық процессор микросхемасының ішінде орналасқан және оның өткізу қабілеттілігі өте жоғары болады. Ортақ жады ағын блогындағы барлық ағындар үшін қолжетімді және деректер алмасуды тиімді ұйымдастыруға мүмкіндік береді.

Ортақ жады бірнеше тәуелсіз жад банктеріне (memory banks) бөлінеді. Әр банкке бір уақытта қол жеткізуге болады, сондықтан әртүрлі банктердегі адрестерге бір мезгілде қол жеткізу жоғары өткізу қабілеттілігін қамтамасыз етеді. Алайда егер бірнеше ағын бір банктегі адрестерге бір уақытта қол жеткізсе, онда bank conflict деп аталатын жағдай туындайды. Мұндай жағдайда жадқа қол жеткізу операциялары тізбектеліп орындалады, бұл жалпы өнімділікті төмендетуі мүмкін.

Сондықтан CUDA бағдарламаларын әзірлеу кезінде жад адрестерінің

банктер бойынша таралуын ескеріп, bank conflict жағдайларын азайту маңызды болып табылады.

Тұрақты жады (Constant memory)

CUDA архитектурасында тұрақты жады (constant memory) тек оқу режимінде қолданылатын арнайы жад кеңістігі болып табылады. Бұл жад құрылғының ғаламдық жадында орналасады, бірақ оған қол жеткізу арнайы тұрақты кэш (constant cache) арқылы жүзеге асырылады.

Тұрақты жады барлық ағындар үшін ортақ болып табылады және әдетте өзгермейтін деректерді сақтау үшін қолданылады. Мұндай деректерге мысал ретінде коэффициенттер, кестелер немесе есептеу барысында өзгермейтін параметрлер жатады.

Егер бір warp ішіндегі барлық ағындар тұрақты жадтың бірдей адресіне жүгінсе, онда деректер кэштен бір рет оқылып, барлық ағындарға бір уақытта таратылады. Мұндай жағдайда жадқа қол жеткізу өте тиімді орындалады.

Ал егер warp ішіндегі ағындар тұрақты жадтың әртүрлі адрестеріне жүгінсе, онда бастапқы сұраныс бірнеше жеке сұраныстарға бөлінеді. Нәтижесінде әрбір адреске бөлек жад операциясы орындалады, бұл жадқа қол жеткізу уақытын арттыруы мүмкін.

Егер деректер constant cache ішінде болса (cache hit), онда жадқа қол жеткізу тұрақты кэштің өткізу қабілеттілігі арқылы жүзеге асырылады және өте жылдам орындалады. Ал егер деректер кэште болмаса (cache miss), онда олар құрылғының ғаламдық жадынан оқылады, бұл жағдайда қол жеткізу жылдамдығы төмен болады.

Тұрақты жады көбінесе барлық ағындар бірдей деректерді пайдаланатын жағдайларда тиімді қолданылады. Сондықтан CUDA бағдарламаларын оңтайландыру кезінде тұрақты жадты дұрыс пайдалану жалпы есептеу өнімділігін арттыруға мүмкіндік береді [22].

1.4 Мультипроцессорлар

Қазіргі заманғы есептеу жүйелерінде көп ядролы процессорлар мен көп процессорлы графикалық үдеткіштердің кеңінен таралуы параллель есептеулердің маңыздылығын айқын көрсетеді. Есептеу құрылғыларының өнімділігі ұзақ уақыт бойы Мур заңына сәйкес дамып келді, яғни транзисторлар санының өсуі есептеу қуатының артуына әкелді. Алайда қазіргі кезеңде өнімділікті арттырудың негізгі жолдарының бірі параллель есептеу әдістерін қолдану болып табылады. Осыған байланысты графикалық процессорлар (GPU) жоғары өнімді есептеулерді жүзеге асыруда маңызды орын алады.

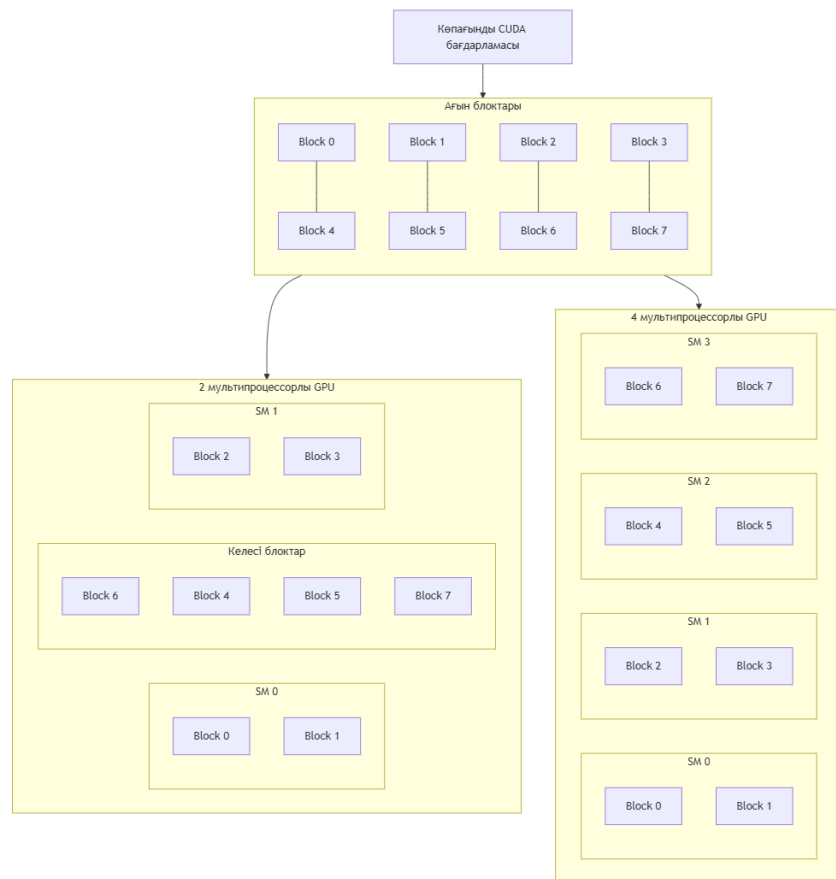
CUDA (Compute Unified Device Architecture) технологиясы NVIDIA компаниясы әзірлеген параллель есептеу платформасы болып табылады. Бұл технология графикалық процессорлардың есептеу мүмкіндіктерін жалпы мақсаттағы есептерді шешу үшін пайдалануға мүмкіндік береді. CUDA бағдарламалау моделі бағдарламашыларға параллель алгоритмдерді құруды жеңілдететін бірнеше негізгі абстракцияларды ұсынады [23].

CUDA бағдарламалау моделінің негізгі элементтері:

- ағындар (threads);

- ағын блоктары (thread blocks);
- ағындар торы (grid);
- ортақ жад (shared memory);
- синхронизация механизмдері.

Бұл абстракциялар бағдарламашыларға есептеу тапсырмаларын тәуелсіз ішкі бөліктерге бөліп, оларды параллель орындауға мүмкіндік береді. Әдетте есептеу міндеті көптеген тәуелсіз операцияларға бөлінеді және бұл операциялар параллель түрде көптеген ағындар арқылы орындалады.



Сурет 5 – CUDA мультипроцессорлық моделі

CUDA бағдарламасында барлық ағындар ағын блоктарына (thread blocks) біріктіріледі. Әрбір блок белгілі бір есептің ішкі бөлігін орындайды және блок ішіндегі ағындар ортақ жадты пайдалана отырып бір-бірімен әрекеттесе алады. Бірнеше блоктар бірігіп ағындар торын (grid) құрайды. Тордағы блоктар бір-бірінен тәуелсіз орындалатындықтан, оларды графикалық процессордағы әртүрлі мультипроцессорларда орындауға болады.

CUDA архитектурасының негізінде ағындық мультипроцессорлар (Streaming Multiprocessor, SM) орналасқан. GPU құрылғысы бірнеше осындай мультипроцессорлардан тұрады. Әрбір мультипроцессор көптеген есептеу ядроларын, регистрлерді, ортақ жадты және басқару блоктарын қамтиды, 5-суретте көрсетілген.

CUDA ядросы (kernel) іске қосылған кезде ағын блоктары автоматты түрде қолжетімді мультипроцессорлар арасында бөлінеді. Әрбір ағын блогы бір мультипроцессорда орындалады, бірақ бір мультипроцессор бір уақытта бірнеше ағын блогын орындай алады. Егер бір блоктың орындалуы аяқталса, оның орнына жаңа блок жоспарланып орындалады. Бұл механизм GPU ресурстарын тиімді пайдалануға мүмкіндік береді.

Мультипроцессор жүздеген немесе мыңдаған ағындарды бір уақытта басқаруға қабілетті. Мұны іске асыру үшін NVIDIA GPU архитектурасында SIMT (Single Instruction Multiple Thread) орындау моделі қолданылады. SIMT архитектурасы бір нұсқаулықты бірнеше ағынның бір уақытта орындауын қамтамасыз етеді.

SIMT моделінде ағындар warp деп аталатын топтарға біріктіріледі. Әдетте бір warp 32 ағыннан тұрады. Warp ішіндегі барлық ағындар бір уақытта бірдей нұсқаулықты орындайды. Бұл тәсіл аппараттық деңгейде параллель есептеулерді тиімді ұйымдастыруға мүмкіндік береді.

Алайда warp ішіндегі ағындар әртүрлі шарттарды орындаған жағдайда branch divergence деп аталатын жағдай туындайды. Мысалы, егер ағындар шартты оператор нәтижесінде әртүрлі орындалу жолдарын таңдаса, онда әрбір тармақ кезекпен орындалады. Бұл жағдайда бір уақытта тек белгілі бір тармаққа тиесілі ағындар ғана белсенді болады, ал қалғандары уақытша тоқтатылады. Нәтижесінде есептеу өнімділігі төмендеуі мүмкін.

SIMT архитектурасы белгілі бір деңгейде SIMD (Single Instruction Multiple Data) архитектурасына ұқсас. SIMD архитектурасында бір нұсқаулық бірнеше деректер элементтеріне қолданылады. Ал SIMT архитектурасында әрбір ағын жеке бағдарламалық контекстке ие болады және өз деректерін өңдей алады. Бұл тәсіл бағдарламалаушыларға икемдірек параллель алгоритмдер құруға мүмкіндік береді.

GPU архитектурасының тағы бір маңызды ерекшелігі – аппараттық көпағындылық (hardware multithreading). Әрбір мультипроцессор бірнеше warp-ты бір уақытта басқаруға қабілетті. Егер бір warp жадқа қол жеткізуді күтіп тұрса, онда жоспарлаушы (scheduler) басқа дайын warp-ты орындауға жібереді. Бұл тәсіл жадқа қол жеткізу кідірістерін жасырып, есептеу ресурстарын тиімді пайдалануға мүмкіндік береді.

Warp жоспарлаушысы әр такт сайын орындалуға дайын warp-ты таңдап, оған сәйкес нұсқаулықты орындауға жібереді. Осылайша мультипроцессор есептеу құрылғыларын үздіксіз жүктеп, жоғары өнімділікті қамтамасыз етеді.

Әрбір мультипроцессорда орындалатын ағындардың жалпы саны бірнеше факторларға байланысты:

- мультипроцессордағы регистрлер саны;
- ортақ жад көлемі;
- бір ядро пайдаланатын ресурстар көлемі;
- GPU архитектурасының ерекшеліктері.

Бұл параметрлер occupancy деп аталатын көрсеткішке әсер етеді. Occupancy – мультипроцессордағы белсенді warp-тардың максималды мүмкін

санына қатынасын сипаттайтын көрсеткіш. Осымен жоғары болған сайын GPU есептеу ресурстары тиімдірек пайдаланылады.

CUDA архитектурасының осындай ұйымдастырылуы графикалық процессорлардың мыңдаған ағындарды параллель орындауына мүмкіндік береді. Нәтижесінде GPU архитектурасы үлкен көлемдегі деректермен жұмыс істейтін есептер үшін өте тиімді болып табылады.

1.5 CUDA технологиясының бағдарламалау моделі

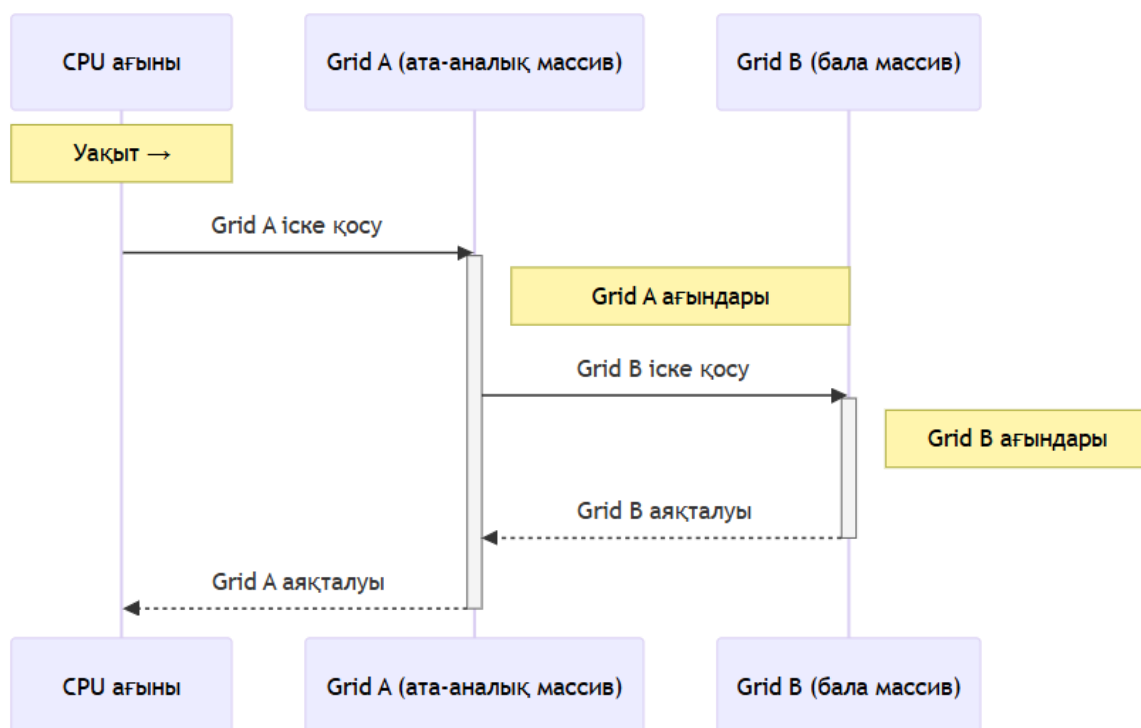
CUDA орындау моделі ағын примитивтеріне, ағын блоктарына және ядро функциялары бар торларға негізделген. Бұл модель бағдарламаның ағын блогында және торда бөлек ағындар арқылы параллель орындалуын қамтамасыз етеді. Ядро функциясы шақырылған кезде тордың параметрлері CUDA-да арнайы синтаксис арқылы анықталатын іске қосу конфигурациясы көмегімен сипатталады.

CUDA технологиясында динамикалық параллелизмді қолдау құрылғыда орындалатын ағындарға жаңа торларды конфигурациялауға, іске қосуға және оларды синхрондауға мүмкіндік береді.

Жаңа торды іске қосатын құрылғы ағыны ата-аналық торға (parent grid) жатады, ал осы шақыру арқылы құрылған тор бала тор (child grid) деп аталады.

Бала торларын іске қосу және орындау иерархиялық түрде ұйымдастырылады. Ата-аналық тор оның ағындары іске қосқан барлық бала торлар толық орындалмайынша аяқталған болып есептелмейді. Шақыру ағындары бала торлармен нақты синхрондалмаған жағдайда да, CUDA орындау ортасы ата-аналық және бала торлар арасында жасырын синхронизацияны қамтамасыз етеді.

CUDA жұмыс уақыты ортасы хостта да, құрылғыда да ядроларды іске қосуға, олардың аяқталуын күтуге және ағындар мен оқиғалар арқылы іске қосулар арасындағы тәуелділіктерді басқаруға арналған API ұсынады. Хост жүйесінде ағындар мен оқиғаларға қатысты бастапқы күй және CUDA примитивтері бір процестегі барлық ағындар арасында ортақ пайдаланылады. Алайда әртүрлі процестер тәуелсіз жұмыс істейді және CUDA объектілерін бөлісе алмайды. CUDA бағдарламалау моделінің құрылымы 6 суретте көрсетілген [24].



Сурет 6 – CUDA бағдарламалау моделі

Құрылғы деңгейінде де ұқсас иерархиялық құрылым қолданылады. Жұмыс істеп тұрған ядролар мен CUDA объектілері бір ағын блогындағы барлық ағындарға көрінеді, алайда әртүрлі ағын блоктары бір-бірінен тәуелсіз жұмыс істейді. Мысалы, белгілі бір ағын бір объектіні құрып, оны сол блоктағы басқа ағындар пайдалана алады, бірақ оны басқа блоктағы ағындар пайдалана алмайды.

CUDA орындау операцияларының барлығы, соның ішінде ядроны іске қосу, ағын блогының барлық ағындарына көрінеді. Бұл ата-аналық тордағы шақыру ағыны сол ағыннан басталған торлармен, сол блоктағы басқа ағындармен немесе сол блокта жасалған басқа ағындармен синхрондала алатынын білдіреді.

Ағын блогының орындалуы блоктағы барлық ағындар басталып, өз жұмысын аяқтағаннан кейін ғана аяқталған болып есептеледі. Егер блоктағы барлық ағындар барлық іске қосу операциялары басталғанға дейін аяқталса, онда синхронизация автоматты түрде орындалады.

CUDA ағындары мен оқиғалары әртүрлі торлар арасындағы тәуелділіктерді басқаруға мүмкіндік береді. Бір ағын ішінде іске қосылған торлар ретімен орындалады, ал оқиғалар бірнеше ағын арасындағы тәуелділіктерді ұйымдастыру үшін қолданылады.

CUDA құрылғысында ядроларды іске қосу тәртібі CUDA ағындарының тізбекті семантикасына сәйкес жүзеге асырылады. Бір ағын блогындағы ядролар бір ағын ішінде ретімен іске қосылады және ретімен орындалады. Бір блоктағы бірнеше ағын бір ағын контекстінде жұмыс істейтіндіктен, орындалу тәртібі блок ішіндегі жоспарлауға байланысты болады. Бұл тәртіпті syncthreads сияқты синхронизация примитивтері арқылы басқаруға болады.

Динамикалық параллелизм бағдарламаның ішінде параллельділікті кеңейтуге мүмкіндік береді. Дегенмен CUDA жұмыс уақыты ортасы әртүрлі ағын блоктары арасында міндетті синхрондау кепілдіктерін бермейді. Сондықтан әртүрлі ағын блоктарының орындалу тәртібіне тәуелді болу қауіпсіз емес.

1.5.1 Хост процессорында CUDA бағдарламаларын ұйымдастыру

Графикалық процессор жедел жадыға тікелей қол жеткізе алмайтындықтан, бағдарламашы есептеу ядросын іске қосу алдында барлық қажетті деректердің GPU жадында орналасуын қамтамасыз етуі тиіс. Осы мақсатта CUDA бағдарламалау ортасында арнайы жадты басқару функциялары қолданылады. Олардың негізгілері: `cudaMalloc`, `cudaMemcpy` және `cudaFree`. Бұл функциялар стандартты `malloc`, `memcpy` және `free` функцияларына ұқсас, бірақ олар графикалық процессордың жадында жұмыс істейді.

CUDA бағдарламасында GPU жадысын бөлу үшін `cudaMalloc` функциясы қолданылады. Бұл функция GPU құрылғысының жадында айнымалыға қажетті көлемдегі жадыны резервтеуге мүмкіндік береді.

GPU мен CPU арасындағы деректер алмасу `cudaMemcpy` функциясы арқылы орындалады. Бұл функция деректерді хост жадынан құрылғы жадысына немесе керісінше көшіруге мүмкіндік береді.

CUDA бағдарламасында есептеулер ядро функциясы (`kernel`) арқылы орындалады. Ядроны іске қосу кезінде тор (`grid`) және блок (`block`) өлшемдері көрсетіледі. Бұл параметрлер GPU-дағы параллель есептеулердің ұйымдастырылуын анықтайды.

Блоктағы ағындар санын дұрыс таңдау өте маңызды. Егер блоктағы ағындар саны тым аз болса, GPU есептеу ресурстары толық пайдаланылмайды. Ал ағындар саны тым көп болған жағдайда регистрлер жетіспеуі мүмкін, нәтижесінде деректер локальды жадқа орналастырылады. Бұл ядроның орындалу уақытын ұлғайтады.

Сондықтан CUDA бағдарламаларын жобалау кезінде тор мен блок өлшемдерінің тиімді конфигурациясын таңдау маңызды болып табылады.

1.5.2 Графикалық процессорда бағдарламаны орындау моделі

CUDA бағдарламалау моделінде функциялардың орындалу орнын анықтау үшін арнайы функция квалификаторлары қолданылады. Бұл квалификаторлар функцияның хостта немесе құрылғыда орындалатынын анықтайды.

CUDA-да ең жиі қолданылатын квалификаторлардың бірі **global** болып табылады. Бұл квалификатор ядро функциясын жариялау үшін қолданылады. Мұндай функция GPU құрылғысында орындалады және CPU-дан шақырылады. Ядро функциясының мысалы (11 сурет) көрсетілген.

Бұл кодта әрбір ағын өз индексін есептеп, сәйкес массив элементін өңдейді. Мұндай тәсіл CUDA-да деректерді параллель өңдеуге мүмкіндік береді.

CUDA-да тағы екі маңызды квалификатор қолданылады: **device** және **host**. `__device__` квалификаторы құрылғыда орындалатын және тек құрылғыдан шақырылатын функцияны жариялайды. Ал `__host__` квалификаторы хостта орындалатын функцияларды анықтайды.

CUDA бағдарламалау моделінде айнымалылардың орналасуын анықтайтын арнайы жад квалификаторлары да бар. Олардың бірі **shared** болып табылады. Бұл квалификатор блок ішіндегі барлық ағындарға ортақ жад аймағын жариялау үшін қолданылады.

Бұл массив блок ішіндегі барлық ағындар үшін қолжетімді болады және ағындар арасында деректер алмасуды жылдам орындауға мүмкіндік береді.

CUDA-да сондай-ақ `constant` квалификаторы қолданылады. Бұл квалификатор тұрақты жад аймағында сақталатын айнымалыларды жариялауға мүмкіндік береді. Тұрақты жад GPU-да кэш арқылы тиімді қолжетімді болғандықтан, жиі қолданылатын тұрақты параметрлерді сақтау үшін пайдаланылады.

Сонымен қатар CUDA компиляторы `restrict` көрсеткіштерін қолдайды. Бұл механизм компиляторға көрсеткіштердің бір жад аймағын пайдаланбайтынын көрсетуге мүмкіндік береді. Мұндай ақпарат компиляторға кодты тиімдірек оңтайландыруға көмектеседі.

Осылайша CUDA бағдарламалау моделінде хост пен құрылғы арасындағы өзара әрекеттесу арнайы API функциялары арқылы жүзеге асырылады, ал GPU-да параллель есептеулер ағындар, блоктар және торлар иерархиясы арқылы ұйымдастырылады [25].

1.6 Жану процестерін модельдеуде параллель есептеулерді қолдану

Жану процестерін модельдеу қазіргі заманғы есептеу гидродинамикасы мен химиялық кинетика салаларындағы күрделі ғылыми есептердің бірі болып табылады. Жану құбылыстары химиялық реакциялар, масса алмасу, жылу алмасу және газ динамикасы сияқты өзара байланысты физикалық-химиялық процестердің жиынтығымен сипатталады. Мұндай процестерді сипаттайтын математикалық модельдер әдетте дифференциалдық теңдеулер жүйесіне негізделеді. Бұл жүйеге массаның сақталу теңдеуі, импульс сақталу теңдеуі, энергия теңдеуі және химиялық реакциялар кинетикасын сипаттайтын теңдеулер кіреді. Сондықтан жану процестерін сандық модельдеу үлкен көлемдегі есептеулерді талап етеді.

Жану процестерін есептеу кезінде кеңістіктік есептеу аймағы торларға (grid) бөлінеді. Әрбір тор ұяшығында температура, қысым, жылдамдық және химиялық компоненттердің концентрациялары анықталады. Сонымен қатар, әрбір химиялық реакцияның жылдамдығы есептеледі. Егер химиялық механизмде жүздеген реакциялар болса, онда есептеу күрделілігі айтарлықтай өседі. Мұндай жағдайда есептеу уақытын қысқарту үшін жоғары өнімді есептеу технологияларын қолдану қажеттілігі туындайды.

Қазіргі уақытта жоғары өнімді есептеулерде параллель есептеу әдістері кеңінен қолданылады. Параллель есептеулердің негізгі идеясы – есептеу тапсырмасын бірнеше тәуелсіз бөліктерге бөліп, оларды бір уақытта бірнеше есептеу ядроларында орындау. Мұндай тәсіл есептеу уақытын айтарлықтай қысқартуға мүмкіндік береді.

1.7 Серверлік есептеу технологиялары және ғылыми есептеулер

Ақпараттық технологиялардың қарқынды дамуы есептеу ресурстарын ұйымдастырудың жаңа тәсілдерінің пайда болуына алып келді. Соңғы жылдары ғылыми есептеулерді орындау үшін кеңінен қолданылатын технологиялардың бірі — бұлтты есептеу (Cloud Computing) технологиясы болып табылады. Бұлтты есептеу пайдаланушыларға есептеу ресурстарын, деректерді сақтау жүйелерін және бағдарламалық қызметтерді интернет желісі арқылы қашықтан пайдалануға мүмкіндік береді.

Дәстүрлі есептеу жүйелерінде ғылыми есептеулер жергілікті серверлерде немесе арнайы есептеу кластерлерінде орындалады. Мұндай жүйелерді орнату және қолдау үлкен қаржылық шығындарды талап етеді. Сонымен қатар есептеу ресурстарын кеңейту үшін қосымша аппараттық құралдарды сатып алу қажет болады. Бұлтты есептеу технологиялары осы мәселелерді шешуге мүмкіндік береді, себебі есептеу ресурстары виртуалды ортада ұйымдастырылып, пайдаланушыларға қажет болған кезде ғана ұсынылады.

Бұлтты есептеу технологиялары әдетте үш негізгі қызмет көрсету моделіне бөлінеді:

Infrastructure as a Service (IaaS) – инфрақұрылымды қызмет ретінде ұсыну. Бұл модельде пайдаланушылар виртуалды серверлерді, деректер сақтау жүйелерін және желілік ресурстарды қолдана алады.

Platform as a Service (PaaS) – платформаны қызмет ретінде ұсыну. Бұл жағдайда пайдаланушылар бағдарламалық қамтамасыз етуді әзірлеуге арналған дайын бағдарламалық орталарды қолданады.

Software as a Service (SaaS) – бағдарламалық қамтамасыз етуді қызмет ретінде ұсыну. Бұл модельде бағдарламалық жүйелер интернет арқылы пайдаланушыларға дайын күйінде ұсынылады.

Ғылыми есептеулерде көбінесе IaaS моделі қолданылады, себебі зерттеушілерге үлкен есептеу ресурстары мен арнайы бағдарламалық орталарды қолдану қажет болады.

Бұлтты есептеу технологияларының негізгі артықшылықтарының бірі — **масштабталу мүмкіндігі** болып табылады. Пайдаланушылар есептеу ресурстарын есептің күрделілігіне байланысты динамикалық түрде арттыра немесе азайта алады. Бұл ғылыми зерттеулер жүргізу кезінде есептеу ресурстарын тиімді пайдалануға мүмкіндік береді.

Сонымен қатар бұлтты есептеу технологиялары есептеу ресурстарының қолжетімділігін арттырады. Пайдаланушылар есептеу жүйелеріне интернет арқылы кез келген жерден қол жеткізе алады. Бұл ғылыми зерттеулер жүргізу кезінде зерттеушілердің жұмысын жеңілдетеді.

Қазіргі уақытта ғылыми есептеулерді орындау үшін Amazon Web Services (AWS), Microsoft Azure және Google Cloud Platform сияқты бұлтты платформалар кеңінен қолданылады. Бұл платформалар жоғары өнімді есептеулерді орындау үшін арнайы есептеу ресурстарын, соның ішінде көп ядролы процессорлар мен графикалық процессорларды ұсынады.

Қазіргі заманғы ғылыми есептеулерде үлкен көлемдегі деректерді өңдеу және күрделі математикалық модельдерді есептеу үшін графикалық процессорлар (GPU) кеңінен қолданылады. GPU архитектурасы параллель есептеулерді тиімді орындауға мүмкіндік береді, себебі олар мындаған есептеу ядроларынан тұрады.

Бұлтты есептеу технологиялары GPU ресурстарын виртуалды ортада пайдалануға мүмкіндік береді. Мұндай жүйелерде пайдаланушылар арнайы GPU серверлерін жалға алып, күрделі ғылыми есептерді орындау үшін қолдана алады.

Cloud + GPU есептеу архитектурасында есептеу жүйесі бірнеше негізгі компоненттерден тұрады:

- пайдаланушы интерфейсі
- бұлтты басқару жүйесі
- есептеу серверлері
- GPU есептеу түйіндері
- деректер сақтау жүйелері

Пайдаланушы есептеу тапсырмасын бұлтты жүйеге жібереді. Содан кейін тапсырма басқару жүйесі арқылы есептеу түйіндеріне таратылады. Егер есептеу параллель алгоритмдерді қолданса, онда есептеу міндеттері бірнеше GPU арасында бөлініп орындалады.

Бұл тәсіл үлкен көлемдегі ғылыми есептерді орындау кезінде есептеу уақытын айтарлықтай қысқартуға мүмкіндік береді. Мысалы, күрделі химиялық кинетика теңдеулерін есептеу кезінде GPU архитектурасын қолдану CPU-мен салыстырғанда бірнеше есе жоғары өнімділік береді.

Жану процестерін модельдеу есептерінде GPU қолдану әсіресе тиімді. Себебі мұндай есептерде әрбір есептеу торының ұяшығында химиялық реакциялар жылдамдығы есептеледі. Бұл операциялар бір-бірінен тәуелсіз болғандықтан, оларды параллель түрде орындауға болады.

Бұлтты GPU есептеу жүйелерін қолдану ғылыми есептеулердің икемділігін арттырады. Зерттеушілер қажет болған жағдайда бірнеше GPU серверлерін пайдаланып, күрделі есептерді қысқа уақыт ішінде орындай алады.

Жоғары өнімді есептеу жүйелері (High Performance Computing — HPC) ғылыми зерттеулерде кеңінен қолданылады. HPC жүйелері әдетте арнайы есептеу кластерлерінен тұрады және үлкен көлемдегі ғылыми есептерді орындауға арналған.

HPC жүйелерінің негізгі артықшылықтары — жоғары өнімділік және тұрақты есептеу ортасы. Мұндай жүйелерде есептеу түйіндері жоғары жылдамдықтағы желілер арқылы біріктіріледі және есептеу міндеттері кластер түйіндері арасында тиімді бөлінеді.

Алайда HPC жүйелерін құру және қолдау үлкен қаржылық шығындарды талап етеді. Сонымен қатар есептеу ресурстарын кеңейту үшін жаңа аппараттық құралдарды орнату қажет.

Бұлтты есептеу технологиялары HPC жүйелеріне балама ретінде қарастырылады. Бұлтты есептеу жүйелерінде пайдаланушылар есептеу ресурстарын виртуалды серверлер арқылы пайдалана алады.

НРС және Cloud есептеу жүйелерінің негізгі айырмашылықтары 1 кестеде көрсетілген.

1 кесте – НРС және Cloud есептеу жүйелерін салыстыру

Параметр	НРС жүйелері	Cloud жүйелері
Инфрақұрылым	жергілікті кластер	виртуалды серверлер
Масштабталу	шектеулі	динамикалық
Құрылу құны	өте жоғары	төмен
Ресурстарға қолжетімділік	жергілікті	интернет арқылы
Икемділік	төмен	жоғары

НРС жүйелері тұрақты ғылыми есептеулер үшін тиімді болса, бұлтты есептеу жүйелері икемділігі жоғары және ресурстарды жылдам масштабтауға мүмкіндік береді.

Қазіргі уақытта көптеген ғылыми орталықтар НРС және Cloud технологияларын біріктіріп қолдануда. Мұндай гибриді есептеу жүйелері ғылыми есептеулерді тиімді орындауға мүмкіндік береді.

Жану процестерін модельдеу есептерінде бұлтты есептеу технологияларын қолдану есептеу ресурстарын тиімді пайдалануға және есептеу уақытын қысқартуға мүмкіндік береді. Сонымен қатар бұлтты жүйелерде есептеу модельдерін автоматты түрде генерациялау және есептеу нәтижелерін қашықтан өңдеу мүмкіндігі бар.

Осылайша бұлтты есептеу технологиялары ғылыми есептеулерді ұйымдастырудың тиімді құралдарының бірі болып табылады. Cloud + GPU архитектурасын қолдану жоғары өнімді есептеу жүйелерін құруға және күрделі ғылыми модельдерді тиімді есептеуге мүмкіндік береді [26].

1.8 Жану модельдерін модельдеуге қойылатын талаптар

Жану процестерін модельдеу қазіргі заманғы ғылыми зерттеулерде маңызды бағыттардың бірі болып табылады. Жану құбылыстары химиялық кинетика, газ динамикасы, жылу алмасу және масса алмасу сияқты күрделі физикалық-химиялық процестердің өзара байланысымен сипатталады. Мұндай процестерді сандық модельдеу үшін көп компонентті химиялық реакциялар жүйесін және дифференциалдық теңдеулердің үлкен жүйесін шешу қажет. Сондықтан жану модельдерін құру және оларды есептеу жүйелерінде қолдану күрделі есептеу міндеті болып табылады.

Жану процестерін сипаттайтын математикалық модельдер жүздеген химиялық реакциялардан және ондаған химиялық компоненттерден тұруы мүмкін. Мұндай модельдерді қолмен құру көп уақытты талап етеді және қателіктердің пайда болу ықтималдығы жоғары болады. Осы себепті жану модельдерін **автоматты түрде генерациялайтын ақпараттық-есептеу жүйелерін** құру қажеттілігі туындайды. Мұндай жүйелер пайдаланушы енгізген бастапқы параметрлер негізінде жану модельдерін автоматты түрде қалыптастырып, оларды сандық есептеулер үшін дайындауы тиіс.

Жану модельдерін автоматты генерациялау жүйелерін жобалау кезінде бірқатар маңызды талаптарды ескеру қажет. Бұл талаптарды бірнеше негізгі топтарға бөлуге болады: функционалдық талаптар, архитектуралық талаптар, есептеу тиімділігіне қойылатын талаптар, деректерді өңдеу талаптары, пайдаланушы интерфейсіне қойылатын талаптар және жүйенің сенімділігі мен қауіпсіздігіне қойылатын талаптар.

Функционалдық талаптар

Жану модельдерін автоматты генерациялау жүйесінің негізгі функциясы — пайдаланушы енгізген бастапқы параметрлер негізінде жану моделін қалыптастыру болып табылады. Жүйе пайдаланушыдан жанғыш қоспаның құрамын, бастапқы температураны, қысымды және реакциялық аймақтың геометриялық параметрлерін қабылдай алуы тиіс.

Сонымен қатар жүйе химиялық реакциялар механизмін автоматты түрде құра алуы қажет. Бұл кезеңде химиялық компоненттердің өзара әрекеттесуін сипаттайтын реакциялар тізімі қалыптастырылады. Әрбір реакция үшін стехиометриялық коэффициенттер, реакция жылдамдығының параметрлері және термодинамикалық сипаттамалар анықталуы тиіс.

Жүйе термодинамикалық қасиеттерді есептеу функцияларын қолдауы қажет. Бұл қасиеттерге заттардың энтальпиясы, энтропиясы, жылусыйымдылығы және Гиббс энергиясы жатады. Мұндай параметрлер химиялық реакциялардың тепе-теңдік күйін және реакция жылдамдықтарын есептеу үшін қажет.

Жүйенің тағы бір маңызды функциясы — алынған химиялық модельді сандық есептеулерге бейімдеу болып табылады. Бұл кезеңде химиялық кинетика теңдеулері сандық алгоритмдер арқылы шешуге дайындалады. Мұндай алгоритмдерге уақыт бойынша интеграциялау әдістері, дифференциалдық теңдеулерді шешу әдістері және тұрақтылықты қамтамасыз ететін сандық схемалар жатады.

Сонымен қатар жүйе есептеу нәтижелерін өңдеу және талдау құралдарын қамтамасыз етуі тиіс. Бұл пайдаланушыға жану процесінің динамикасын, температура өрісін, концентрациялардың өзгерісін және реакция жылдамдықтарын талдауға мүмкіндік береді.

Архитектуралық талаптар

Жану модельдерін автоматты генерациялау жүйесі модульдік архитектура принципі бойынша құрылуы тиіс. Модульдік архитектура жүйенің әрбір компонентін тәуелсіз түрде дамытуға және жаңартуға мүмкіндік береді.

Жүйенің негізгі модульдері келесідей болуы мүмкін:

- бастапқы параметрлерді енгізу модулі
- химиялық механизмдерді генерациялау модулі
- термодинамикалық параметрлерді есептеу модулі
- сандық есептеу алгоритмдерін орындау модулі
- нәтижелерді визуализациялау модулі
- деректерді сақтау және басқару модулі

Мұндай архитектура жүйенің масштабталуын және икемділігін арттырады. Сонымен қатар жүйені әртүрлі ғылыми есептерге бейімдеуге мүмкіндік береді.

Есептеу тиімділігіне қойылатын талаптар

Жану процестерін модельдеу есептері үлкен көлемдегі есептеулерді талап етеді. Сондықтан автоматты генерациялау жүйесі жоғары өнімді есептеу технологияларын қолдауы тиіс. Мұндай технологияларға параллель есептеулер, көп ядролы процессорларды қолдану және графикалық процессорларды пайдалану жатады.

GPU негізіндегі есептеу технологиялары жану модельдерін есептеу кезінде айтарлықтай тиімді болып табылады. Себебі GPU архитектурасы мыңдаған параллель есептеу ядроларынан тұрады және үлкен көлемдегі деректерді параллель өңдеуге мүмкіндік береді.

Сонымен қатар жүйе бұлтты есептеу технологияларын қолдауы тиіс. Бұл пайдаланушыларға қашықтағы есептеу ресурстарын пайдаланып, күрделі ғылыми есептерді орындауға мүмкіндік береді.

Жүйе есептеу ресурстарын тиімді пайдалануы қажет. Бұл үшін параллель алгоритмдерді қолдану, жады ресурстарын оңтайлы пайдалану және есептеу міндеттерін тиімді жоспарлау маңызды.

Деректерді өңдеу талаптары

Жану модельдерін автоматты генерациялау жүйесі үлкен көлемдегі деректермен жұмыс істей алуы тиіс. Химиялық реакциялар жүйесі көптеген компоненттер мен реакциялардан тұруы мүмкін. Сондықтан жүйеде деректерді сақтау және өңдеу үшін тиімді құрылымдар қолданылуы қажет.

Деректер базасы химиялық компоненттердің қасиеттерін, реакция параметрлерін және термодинамикалық коэффициенттерді сақтауға мүмкіндік беруі тиіс. Сонымен қатар деректерді жылдам іздеу және өңдеу механизмдері қарастырылуы қажет.

Жүйе үлкен көлемдегі есептеу нәтижелерін сақтауға және оларды кейінгі талдау үшін қолдануға мүмкіндік беруі тиіс.

Пайдаланушы интерфейсіне қойылатын талаптар

Жүйе пайдаланушыға ыңғайлы интерфейспен қамтамасыз етілуі тиіс. Пайдаланушы жүйеге бастапқы параметрлерді оңай енгізіп, есептеу процесін бақылап және алынған нәтижелерді талдай алуы қажет.

Пайдаланушы интерфейсі есептеу параметрлерін енгізу, модельді генерациялау, есептеу процесін іске қосу және нәтижелерді визуализациялау функцияларын қамтуы тиіс.

Сонымен қатар интерфейс графикалық визуализация құралдарын қамтуы қажет. Бұл пайдаланушыға температура өрістерін, концентрациялардың өзгерісін және реакция жылдамдықтарын графиктер мен диаграммалар арқылы көруге мүмкіндік береді.

Сенімділік және қауіпсіздік талаптары

Жүйе үлкен көлемдегі есептеулерді орындау кезінде тұрақты жұмыс істеуі тиіс. Сонымен қатар есептеу нәтижелерінің дәлдігі мен сенімділігі қамтамасыз етілуі қажет.

Жүйе пайдаланушы деректерін қорғау механизмдерін қолдауы тиіс. Бұл әсіресе бұлтты есептеу ортасында маңызды, себебі деректер қашықтағы серверлерде сақталады.

Сонымен қатар жүйе қателерді анықтау және өңдеу механизмдерін қамтуы қажет. Бұл есептеу процесі кезінде туындайтын техникалық немесе бағдарламалық қателердің әсерін азайтуға мүмкіндік береді.

Жану модельдерін автоматты генерациялау жүйелерін құру күрделі бағдарламалық-есептеу міндеті болып табылады. Мұндай жүйелер химиялық кинетика модельдерін автоматты түрде қалыптастыруға, сандық есептеулерді тиімді орындауға және ғылыми зерттеулердің өнімділігін арттыруға мүмкіндік береді.

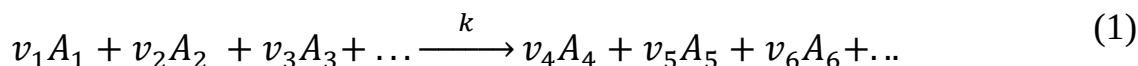
Функционалдық, архитектуралық және техникалық талаптарды дұрыс анықтау жүйенің тиімді жұмыс істеуін қамтамасыз етеді. Сонымен қатар параллель есептеу технологияларын, GPU архитектурасын және бұлтты есептеу жүйелерін қолдану жану процестерін модельдеу есептерін орындаудың жаңа мүмкіндіктерін ашады.

Осы талаптарды ескере отырып құрылған ақпараттық-есептеу жүйелері жану модельдерін автоматты түрде генерациялау және оларды жоғары өнімді есептеу орталарында тиімді қолдану үшін негіз болады[27].

2 ЖОҒАРЫ ӨНІМДІ ЕСЕПТЕУ ЖҮЙЕСІН ҰЙЫМДАСТЫРУ

2.1 Мәселенің математикалық қойылымы

Химиялық реакцияның жылдамдығын, жалпы түрде жазылған, реакция мысалында қарастырайық [28]:



мұндағы, $A_1, A_2, A_3, \dots$ реакцияға қатысатын әр түрлі компоненттерді білдіреді, k - жылдамдық тұрақтысы және $v_1, v_2, v_3, A_1, A_2, A_3, \dots$ компоненттеріне қатысты реакцияның реті. Реакция жылдамдығын келесі түрде жазуға болады:

$$\frac{dC_1}{dt} = -k C_1^{v_1} C_2^{v_2} C_3^{v_3} \dots \quad (2)$$

мұндағы, $C_1, C_2, C_3, \dots$ концентрациялар.

Мұндай теңдеулер химиялық кинетикада әрекет етуші массалар заңына негізделеді және көпкомпонентті реакциялық жүйелерді сандық модельдеу кезінде кеңінен қолданылады [29, 30].

Жеті реакциядан тұратын біршама күрделі реакция жүйесінің әр компоненттеріне теңдеу құрастыру үшін:

- 1) $H + O_2 \rightleftharpoons O + OH$
- 2) $O + H_2 \rightleftharpoons H + OH$
- 3) $OH + H_2 \rightleftharpoons H_2O + H$
- 4) $O + H_2O \rightleftharpoons 2OH$
- 5) $H_2 + M \rightleftharpoons 2H + M$
- 6) $H + OH + M \rightleftharpoons H_2O + M$
- 7) $H + HO_2 \rightleftharpoons H_2 + O_2$

мұндағы: $M = N_2$

$$\frac{dC_k}{dt} = \sum_{i=1}^{N_r} \left(v_{ik}^+ \left[-k_i^+ \prod_{j=1}^{N_c} C_j^{v_{ij}^+} + k_i^- \prod_{j=1}^{N_c} C_j^{v_{ij}^-} \right] - v_{ik}^- \left[-k_i^+ \prod_{j=1}^{N_c} C_j^{v_{ij}^+} + k_i^- \prod_{j=1}^{N_c} C_j^{v_{ij}^-} \right] \right) \quad (3)$$

мұндағы, N_r - реакция саны, N_c - компоненттер саны

$C_1 - H_2, C_2 - O_2, C_3 - H_2O, C_4 - OH, C_5 - H, C_6 - HO_2, C_7 - O, C_8 - N_2$.

Әр компонент үшін сәйкес индекстерді қойып, келесі түрдегі дифференциалдық теңдеулер жүйесін аламыз:

$$\begin{aligned}
\frac{dC_1}{dt} &= -k_2 C_1 C_7 + k_{-2} C_4 C_5 - k_3 C_1 C_4 + k_{-3} C_3 C_5 - k_5 C_1 C_8 \\
&\quad + k_{-5} C_5^2 C_8 + k_7 C_5 C_6 - k_{-7} C_1 C_2 \\
\frac{dC_2}{dt} &= -k_1 C_2 C_5 + k_{-1} C_4 C_7 + k_7 C_5 C_6 - k_{-7} C_1 C_2 \\
\frac{dC_3}{dt} &= k_3 C_1 C_4 - k_{-3} C_3 C_5 - k_4 C_3 C_7 + k_{-4} C_4^2 + k_6 C_4 C_5 C_8 \\
&\quad - k_{-6} C_3 C_8 \\
\frac{dC_4}{dt} &= k_1 C_2 C_5 - k_{-1} C_4 C_7 + k_2 C_1 C_7 - k_{-2} C_4 C_5 - k_3 C_1 C_4 \\
&\quad + k_{-3} C_3 C_5 + 2k_4 C_3 C_7 - 2k_{-4} C_4^2 + 2k_5 C_1 C_8 \\
&\quad - 2k_{-5} C_5^2 C_8 - k_6 C_4 C_5 C_8 + k_{-6} C_3 C_8 \\
\frac{dC_5}{dt} &= -k_1 C_2 C_5 + k_{-1} C_4 C_7 + k_2 C_1 C_7 - k_{-2} C_4 C_5 + k_3 C_1 C_4 \\
&\quad - k_{-3} C_3 C_5 + 2k_5 C_1 C_8 - 2k_{-5} C_5^2 C_8 - k_6 C_4 C_5 C_8 \\
&\quad + k_{-6} C_3 C_8 - k_7 C_5 C_6 + k_{-7} C_1 C_2 \\
\frac{dC_6}{dt} &= -k_7 C_5 C_6 + k_{-7} C_1 C_2 \\
\frac{dC_7}{dt} &= k_1 C_2 C_5 - k_{-1} C_4 C_7 - k_2 C_1 C_7 + k_{-2} C_4 C_5 - k_4 C_3 C_7 \\
&\quad + k_{-4} C_4^2 \\
\frac{dC_8}{dt} &= 0
\end{aligned} \tag{4}$$

Алынған дифференциалдық теңдеулер жүйесі химиялық реакциялар механизміне кіретін компоненттердің уақыт бойынша өзгерісін сипаттайды. Реакциялар саны мен компоненттер саны артқан сайын мұндай жүйелердің өлшемі ұлғайып, оларды сандық шешу үшін арнайы кинетикалық солверлерді қолдану қажеттілігі туындайды [31].

2.2 Таратылған жоғарыөнімділікті есептеу жүйелерінде қолдануға арналған кинетикалық солвер реализациясы

Қысымы тұрақты ортада энергияның сақталуының негізгі теңдеуі:

$$\frac{dT}{dt} = -\frac{1}{\rho c_p} \sum_{k=1}^K H_k \dot{\omega}_k \quad (5)$$

мұндағы, T – ортаның температурасы, ρ – массалық тығыздық, c_p – меншікті жылусыйымдылық, H_k – заттың молярлық энтальпиясы және $\dot{\omega}_k$ – жылдамдық [32].

Стандартты жағдайда заттардың термодинамикалық қасиеті температуралы «идеалды» болып табылады, сондықтан бұл қасиеттер температуралы-тәуелді функциялар сияқты анықталады және қысымы тұрақты ортада бұл заттардың жылусыйымдылығы полиномиалды коэффициенттерге және температураға тәуелді функция ретінде жазылады:

$$\frac{C_{pk}^0}{R} = \sum_{n=1}^N a_{nk} T_k^{(n-1)} \quad (5)$$

мұндағы «0» белгісі заттың 1 атм.қысымға тең қасиетін анықтайды. Идеалды газдар үшін жылусыйымдылық температураға тәуелді болмайды, сондықтан оларға қалыпты жағдайдағы жылусыйымдылық өзекті мағына болып табылады. Берілген температурадағы басқа термодинамикалық қасиеттері жылусыйымдылыққа тәуелді интегралды функция ретінде анықталады.

Қалыпты жағдайдағы заттардың молярлы энтальпиялық күйі төмендегі формуламен анықталады:

$$H_k^0 = \int_{298}^{T_k} C_{pk}^0 dT + H_k^0(298) \quad (7)$$

$$\frac{H_k^0}{RT_k} = \sum_{n=1}^N \frac{a_{nk} T_k^{(n-1)}}{n} + \frac{a_{N+1,k}}{T_k} \quad (7.1)$$

мұндағы, $a_{N+1,k} * R$ параметрі 298K температурада түзілу жылуын білдіреді.

Қалыпты жағдайдағы молярлы энтропия төмендегідей жазылады:

$$S_k^0 = \int_{298}^{T_k} \frac{C_{pk}^0}{T} dT + S_k^0(298) \quad (8)$$

$$\frac{S_k^0}{R} = a_{1k} \ln T_k + \sum_{n=2}^N \frac{a_{nk} T_k^{(n-1)}}{(n-1)} + a_{N+2,k} \quad (8.1)$$

мұндағы, $a_{N+2,k} * R$ интеграция тұрақтысы 298K температурасында берілген заттың энтропиясының мәніне байланысты анықталады.

Жоғарыда берілген теңдеулер (14 теңдеу) химиялық теңдеулерде

пайдаланылатын еркін қатарлы полиномдарға арналған теңдеулер болып табылады. Chemkin программалық кешенінде бұл коэффициенттер температуралық интервалдың екі аралығы үшін екі топқа бөлінеді (төмен және жоғары температура аймағына). Бұл жағдайда берілген заттың термодинамикалық қасиетін анықтау үшін төмендегі теңдеулер пайдаланылады [33,34]:

$$\frac{C_{pk}^0}{R} = a_{1k} + a_{2k}T_k + a_{3k}T_k^2 + a_{4k}T_k^3 + a_{5k}T_k^4 \quad (9)$$

$$\frac{H_k^0}{RT_k} = a_{1k} + \frac{a_{2k}}{2}T_k + \frac{a_{3k}}{3}T_k^2 + \frac{a_{4k}}{4}T_k^3 + \frac{a_{5k}}{5}T_k^4 + \frac{a_{6k}}{6} \quad (10)$$

$$\frac{S_k^0}{R} = a_{1k} \ln(T_k) + a_{2k}T_k + \frac{a_{3k}}{2}T_k^2 + \frac{a_{4k}}{3}T_k^3 + \frac{a_{5k}}{4}T_k^4 + a_{7k} \quad (11)$$

Басқа термодинамикалық қасиеттер C_p^0 , H^0 и S^0 мәндерін пайдалану барысында анықталады. Тұрақты көлемдегі C_{vk}^0 меншікті жылусыйымдылық

$$C_{vk}^0 = C_{pk}^0 - R \quad (12)$$

Ішкі энергия U_k^0

$$U_k^0 = H_k^0 - RT \quad (13)$$

Қалыпты күйдегі Гиббс еркін энергиясы G_k^0

$$G_k^0 = H_k^0 - T_k S_k^0 \quad (14)$$

Қалыпты күйдегі Гельмгольц еркін энергиясы A_k^0

$$A_k^0 = U_k^0 - T_k S_k^0 \quad (15)$$

Қалыпты жағдайдағы идеалды газдардың ішкі энергиясы, жылусыйымдылығы, энтальпиясы және меншікті жылусыйымдылығы осы қасиеттердің нақты мәніне тең.

Реакциялардың жылдамдылығын анықтауға арналған теңдеу

$$\sum_{k=1}^K v'_{ki} \chi_k \Leftrightarrow \sum_{k=1}^K v''_{ki} \chi_k \quad (i = 1, \dots, I) \quad (16)$$

мұндағы v_{ki} бүтін санды стехиометриялық коэффициент және χ_k k –ға тең заттың химиялық таңбасы. v'_{ki} – тура реакцияның жылдамдығы, v''_{ki} – кері реакцияның жылдамдығы. Әдетте, қарапайым реакцияларда үш немесе төрт заттар реакцияға түседі.

k – шы заттың ω_k – жылдамдығының қалыптасуы келесі теңдеуде

анықталады:

$$\dot{\omega}_k = \sum_{i=1}^I v_{ki} q_i \quad (k = 1, \dots, K) \quad (17)$$

мұндағы $v_{ki} = v''_{ki} - v'_{ki}$.

i – i – ші реакциясының жылдамдығының өзгеру реакциясы төмендегі теңдеуде анықталады

$$q_i = k_{fi} \prod_{k=1}^K [X_k]^{v''_{ki}} - k_{ri} \prod_{k=1}^K [X_k]^{v'_{ki}} \quad (18)$$

$[X_k]$ – k – шы заттың мольдік концентрациясы, k_{fi} және k_{ri} тура және кері реакциялардың жылдамдығы.

Тура реакцияның жылдамдығы Аррениустың теңдеуі арқылы анықталады

$$k_{fi} = A_i T^{\beta_i} \exp\left(\frac{-E_i}{R_c T}\right) \quad (19)$$

мұндағы:

R – универсал газ тұрақтысы;

A_i – реакция түріне байланысты анықталатын, температураға тәуелсіз предэкспоненциалдық көбейткіш.

β_i – температуралық экспонент,

E_i – шекті энергия деп сипаттауға болатындай i – ші реакциядағы энергия белсендіруі: шамамен айтқанда, егер соқтығысу энергияның бөлшектері E_i кем болса, онда соқтығысу барысында реакция орындалмайды, ал егер энергия E_i – көбірек болса, реакция орындалады.

Аррениус теңдеуі реакция жылдамдығы тұрақтысының температураға тәуелділігін сипаттайтын негізгі өрнектердің бірі болып табылады. Бұл теңдеу жану процестерінің кинетикалық модельдерінде тура және кері реакция жылдамдықтарын анықтау үшін кеңінен қолданылады [35].

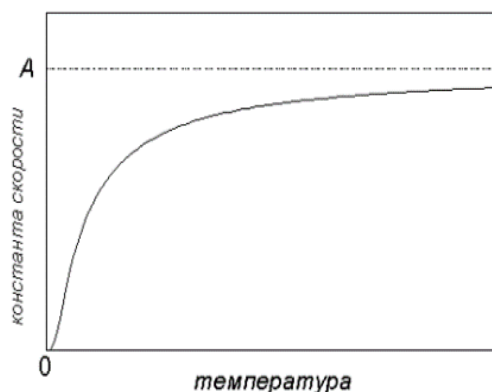
Яғни, егер соқтығысатын бөлшектердің энергиясы E_i -ден төмен болса, онда соқтығысу кезінде реакция жүрмейді. Ал егер бөлшектердің энергиясы E_i -ден жоғары болса, реакция жүреді. Белсендіру энергиясы температураға тәуелді болмайды.

Төмен температурада k_{fi} мәні өте аз болады, сондықтан химиялық реакциялар дерлік жүрмейді. Ал температура жоғарылаған сайын тура реакцияның жылдамдық тұрақтысы артып, шекті мәнге ұмтылады:

$$k_{fi} \rightarrow A_i \quad (20)$$

Барлық бұл молекулалар химиялық түрде белсенді және олардың әрбір соқтығысуы реакцияға алып келетінін көрсетеді.

$k_{fi}(T)$ – графикалық тәуелділігі келесі түрде бейнеленеді:



7 сурет - $k_{fi}(T)$ -графикалық тәуелділігі.

Кері реакцияның жылдамдығы төмендегі формуламен анықталады

$$k_{ri} = \frac{k_{fi}}{K_{ci}} \quad (21)$$

$$K_{ci} = K_{pi} \left(\frac{P_{atm}}{RT} \right)^{\sum_{k=1}^K v_{ki}} \quad (22)$$

мұндағы, K_{pi} – тепе-теңдік коэффициенті келесі теңдеумен анықталады

$$K_{pi} = \exp \left(\frac{\Delta S_i^0}{R} - \frac{\Delta H_i^0}{RT} \right) \quad (23)$$

Δ – i –ші реакцияның реагенттен өнімге айналу барысындағы өзгерісті анықтайды, нақтырақ айтқанда:

$$\frac{\Delta S_i^0}{R} = \sum_{k=1}^K v_{ki} \frac{S_k^0}{R} \quad (24)$$

$$\frac{\Delta H_i^0}{RT} = \sum_{k=1}^K v_{ki} \frac{H_k^0}{RT} \quad (25)$$

Электрондардың қатысуымен өтпелі реакциялар үшін бұл теңдеуде кері реакцияның жылдамдығын табу дұрыс емес. Кейде мұндай қиындықтарды шешуде үшінші денедегі ионизациялау және қайта комбинациялау түріндегі функциялары үшін Саха теңдеуі пайдаланылады.

3 ЖАНУ ПРОЦЕССИН СИПАТТАЙТЫН АҚПАРАТТЫҚ-ЕСЕПТЕУ ЖҮЙЕСІН ҚҰРУ

3.1 Метанның жану механизмі

Алкандар тәрізді көмірсутектердің жану үдерістері өнеркәсіптің көптеген салаларында маңызды орын алады, себебі олар әртүрлі энергетикалық және технологиялық құрылғыларда кеңінен қолданылады. Атап айтқанда, метан мен табиғи газ көлік қозғалтқыштарында, газ айдау қондырғыларының қозғалтқыштарында, жылыту жүйелерінде, өртеу және гидрокрекинг пештерінде негізгі отын түрі ретінде пайдаланылады. Осыған байланысты мұндай жану үдерістерін жоғары дәлдікпен модельдеу өнеркәсіптік тиімділікті арттырумен қатар, қоршаған ортаға түсетін әсерді бағалау және азайту үшін де ерекше маңызға ие.

Мұндай нәтижеге қол жеткізу үшін жүйеде жүретін химиялық түрленулерді сипаттайтын толыққанды фундаменталды кинетикалық модель құру қажет. Бұл модельге реакцияларға қатысатын барлық химиялық бөлшектер, олардың кинетикалық параметрлері, сондай-ақ термодинамикалық және тасымалдау қасиеттері енгізілуі тиіс [36–39]. Әрбір реакция мен әрбір химиялық компонент үшін сәйкес кинетикалық сипаттамалармен қатар, олардың термодинамикалық және көліктік параметрлерін анықтау модельдің сенімділігін қамтамасыз етеді.

Кинетикалық модельдер химиялық үдерістерді сандық тұрғыдан сипаттаудың негізгі құралы болып табылады. Олар қарапайымдандырылған зертханалық жағдайларда алынған эксперименттік деректерге сүйене отырып, практикалық маңызы бар күрделі жүйелердің мінез-құлқын болжауға мүмкіндік береді. Реакция жылдамдығының температураға тәуелділігі, әдетте, Аррениус теңдеуі арқылы сипатталады. Температура артқан сайын көптеген химиялық реакциялардың жүру жылдамдығы да өседі. Бұл әсер негізінен реакция жылдамдығы тұрақтысының өзгеруімен байланысты, ал аталған тәуелділік Аррениус заңында келесі түрде өрнектеледі:

$$k = A * T^n * e^{\frac{E_a}{RT}} \quad (26)$$

мұндағы k — реакция жылдамдығының тұрақтысы, A — предэкспоненциалдық көбейткіш, T — абсолюттік температура, n — температуралық көрсеткіш, E_a — активация энергиясы, R — универсал газ тұрақтысы, ал e — экспоненциалдық функцияның негізі. Бұл теңдеу реакция жылдамдығы тұрақтысының температураға тәуелділігін сипаттайды және температура артқан сайын k мәнінің өсуін көрсетеді.

Химиялық кинетиканың маңызды параметрлерінің бірі — реакция жылдамдығы. Ол тек әрекеттесуші бөлшектердің концентрациясына немесе қысымына және температураға ғана тәуелді емес, сонымен қатар катализаторлардың болуына, реакциялық ыдыстың геометриялық пішіні мен өлшемдеріне, қолданылатын материалдардың қасиеттеріне және қабырға бетінің күйіне де байланысты болады. Жалпы жағдайда реакция жылдамдығы деп

белгілі бір уақыт аралығында көлем бірлігіндегі бастапқы заттардың немесе реакция өнімдерінің мөлшерінің өзгеруін айтады. Бұл шама химиялық үдерістің жүру қарқындылығын сандық тұрғыдан сипаттайды және келесі түрде өрнектеледі:

$$\omega = \pm \frac{1}{V} \frac{dn}{dt} \quad (27)$$

мұндағы, n – моль соны (немесе молекула саны).

Реакция механизмінің толық сипаттамасы химиялық үдерістің молекулалық деңгейде қалай жүзеге асатынын түсіндіруге мүмкіндік береді. Мұндай механизмде қандай химиялық байланыстардың үзіліп, қандай жаңа байланыстардың түзілетіні, реакциялық сатылардың қандай реттілікпен жүретіні және әрбір элементар қадамның жылдамдығына қандай факторлар әсер ететіні қарастырылады.

Нақты реакциялық механизмге қойылатын негізгі талаптар төмендегідей:

- механизм бойынша алынатын қорытынды теңдеу жалпы теңестірілген химиялық теңдеуге сәйкес келуі тиіс;
- механизмнің аралық сатыларына қатысты болжамдар тәжірибелік бақылауларға қайшы келмеуі қажет;
- механизмнен шығатын реакция жылдамдығының теңдеуі эксперименттік деректермен үйлесуі керек;
- егер бірнеше механизм тәжірибелік нәтижелерді бірдей сипаттай алса, олардың дұрыстығын салыстыру және ең қолайлысын таңдау үшін қосымша тәжірибелік зерттеулер жүргізу қажет.

Кинетикалық механизм құрамына реакциялар жиынтығы, олардың жылдамдық коэффициенттері, термодинамикалық сипаттамалар және тасымалдау коэффициенттері кіреді. Реакция жылдамдығын анықтау үшін әртүрлі тәсілдер қолданылады, олардың қатарына эмпирикалық, статистикалық және кванттық-химиялық әдістер жатады.

Химиялық бөлшектердің термодинамикалық қасиеттері, әдетте, температураға тәуелді түрде арнайы кестелерде беріледі. Бұл кестелерде тәжірибелік өлшеулерге негізделген және теориялық есептеулермен толықтырылған термодинамикалық шамалардың мәндері қамтылады. Осы жұмыста пайдаланылған термодинамикалық деректердің негізгі бөлігі [40–42] әдебиеттерінен алынды.

Кинетикалық механизмдегі реакция жылдамдығы коэффициенттерін анықтауға қажетті негізгі параметрлер 7-суретте көрсетілген. Әрбір химиялық реакция үшін Аррениус теңдеуіндегі A — предэкспоненциалдық көбейткіш, ал E_a — активация энергиясы ретінде анықталады. Предэкспоненциалдық көбейткіш әрекеттесуші молекулалардың соқтығысу жиілігін және олардың тиімді соқтығысу ықтималдығын сипаттайды, ал активация энергиясы реакцияның жүруі үшін қажет минималды энергия мөлшерін көрсетеді.

REACTIONS	MOLES	KELVINS	$k_j(T) = A_j T^{n_j} \exp(-E_{a,j} / RT)$		
1			A	n	E_a
H2+CH2(S) = CH3+H			7.230E+13	.0000	.00
2					
CH4+O2 = CH3+HO2			3.970E+13	.0000	28631.22
3					
CH4+C = CH+CH3			5.000E+13	.0000	12085.64
4					
CH4+H = CH3+H2			6.600E+08	1.6000	5420.00
5					
CH4+CH = C2H4+H			3.010E+13	.0000	-199.66
6					
CH4+CH2 = 2CH3			4.300E+12	.0000	5051.72
7					
CH4+CH2(S) = 2CH3			7.000E+13	.0000	.00

Сурет 7 – Реакцияның кинетикалық механизмі

7 және 8-суреттерде реакцияға қатысатын әрбір химиялық бөлшектің термодинамикалық қасиеттерін сипаттайтын деректер қорының форматы ұсынылған. Бұл деректер қоры Sandia деректер базасының құрылымына сәйкес жасалған және оның мазмұны өзгертілмеген күйінде қолданылады. Термодинамикалық мәліметтер құрамында реагенттің атауы, элементтік құрамы, температураның қолданылу аралығы, төмен және жоғары температуралық аймақтарға өту шекарасын анықтайтын ауыспалы температура, сондай-ақ төмен және жоғары температура диапазондарына сәйкес келетін сандық коэффициенттер көрсетіледі.

CH4	121286C	1H	4	G	0300.00	5000.00	1000.00	1
0.01683478E+02	0.10237236E-01	-0.03875128E-04	0.06785585E-08	-0.04503423E-12				2
-0.10080787E+05	0.09623395E+02	0.07787415E+01	0.01747668E+00	-0.02783409E-03				3
0.03049708E-06	-0.12239307E-10	-0.09825229E+05	0.13722195E+02					4
CO	121286C	1O	1	G	0300.00	5000.00	1000.00	1
0.03025078E+02	0.14426885E-02	-0.05630827E-05	0.10185813E-09	-0.06910951E-13				2
-0.14268350E+05	0.06108217E+02	0.03262451E+02	0.15119409E-02	-0.03881755E-04				3
0.05581944E-07	-0.02474951E-10	-0.14310539E+05	0.04848897E+02					4
CO2	121286C	1O	2	G	0300.00	5000.00	1000.00	1
0.04453623E+02	0.03140168E-01	-0.12784105E-05	0.02393996E-08	-0.16690333E-13				2
-0.04896696E+06	-0.09553959E+01	0.02275724E+02	0.09922072E-01	-0.10409113E-04				3
0.06866686E-07	-0.02117280E-10	-0.04837314E+06	0.10188488E+02					4

Сурет 8 – Реагенттердің термодинамикалық қасиетінің негізгі көрінісі

Name	Elements					Temperature range	Junction
N2	0N	2C	0H	0O	0S	0300.00 5000.00 (1000.00)	1
2.9266400E+00	1.4879770E-03	-5.6847603E-07	1.0097040E-10	-6.7533509E-15			2
-922.76760E+00	5.9005290E+00	3.2986770E+00	1.4082399E-03	-3.9632218E-06			3
5.6415148E-09	-2.4418540E-12	-1020.9000E00	3.9503720E+00				4

High temperature coefficients

Low temperature coefficients

Сурет 9 – Реагенттердің термодинамикалық қасиетінің анықтамасы

Кинетикалық механизмнің негізгі құрамдас бөліктерінің бірі — жану үдерісіндегі тасымалдау қасиеттері. 9-суретте тасымалдау қасиеттерінің берілу форматына мысал келтірілген. Бұл мәліметтер құрамына реагенттің атауы кіреді, мұнда реагент атауы термодинамикалық деректердегі сияқты бірегей болуы тиіс. Сонымен қатар геометриялық конфигурацияны сипаттайтын индекс, потенциалдық шұңқырдың тереңдігі, соқтығысу диаметрі, дипольдік момент, поляризациялану шамасы және 298 К температурадағы айналмалы релаксацияның соқтығысу саны көрсетіледі.

Потенциалдық шұңқыр деп бөлшектің потенциалдық энергиясы локальды минимумға ие болатын кеңістік аймағын айтады. Ал дипольдік момент полярлы байланысты сипаттайтын векторлық шама болып табылады және ол келесі өрнекпен анықталады: $|m| = l \cdot q$, мұндағы l — байланыс ұзындығы, ал q — электрондық тығыздықтың ығысуы нәтижесінде пайда болатын атомдардың эффективті заряды. Дипольдік момент векторы оң зарядтан теріс зарядқа қарай бағытталады.

Қазіргі уақытта метанның тотығуын сипаттайтын кеңінен қолданылатын механизмдердің бірі — Беркли университетінің зерттеу тобы әзірлеген GRI-Mech механизмі болып табылады. Бұл механизмнің бірнеше электрондық нұсқалары жарияланған [43–45] және ол бастапқыда метан мен табиғи газдың жану үдерістерін модельдеу үшін жасалған. GRI-Mech 3.0 нұсқасы 325 элементар реакциядан және 53 химиялық компоненттен тұрады. Аталған механизм отынның эквиваленттік коэффициенті 0,1–5 аралығында, қысым 1,0–1000 кПа аралығында және температура 1000–1500 К диапазонында қолдануға арналған.

Сонымен қатар бұл механизм метан жануының кейбір ерекше аспектілерін толық сипаттай алмайды. Мысалы, оның құрамында метанол мен ацетилен компоненттері болғанымен, кейбір тотығу және күйе түзілу үдерістерін дәл анықтауда шектеулер байқалады. Осы кемшіліктерге қарамастан, механизм физика-химиялық деректерге негізделгендіктен және көптеген эксперименттік жұмыстармен салыстырылып тексерілгендіктен, жоғары сенімділікке ие болды [46]. GRI-Mech механизмін жетілдіру жұмыстары 2000 жылдың ақпан айында аяқталғанымен, ол қазіргі кезде де практикалық және ғылыми есептерді шешуде кеңінен қолданылады.

GRI-Mech құрамында көптеген термодинамикалық параметрлер бар. Сонымен қатар механизм эксперименттік жолмен алынған түтіктердегі зат концентрацияларын сипаттауға мүмкіндік береді. Дегенмен, ламинарлы жану жағдайларында, сондай-ақ идеалды араластыру және ығыстыру реакторларындағы кейбір нәтижелерді бұл механизм жеткілікті дәлдікпен сипаттай алмауы мүмкін, бұл 10-суретте көрсетілген.

Метан-сутегі қоспасының жануы да жану кинетикасында кеңінен зерттелген бағыттардың бірі болып табылады. Бұл мәселенің өзектілігі метанның жану қабілетінің салыстырмалы түрде төмен болуымен байланысты. Зерттеулер көрсеткендей, метан-ауа қоспасына сутегіні қосу оның тұтану шектерін кеңейтіп, отынды пайдалану тиімділігін арттыруға және қоршаған ортаға шығарылатын зиянды шығарындылар мөлшерін азайтуға мүмкіндік береді [47].

Алайда кейбір зерттеулерде метан-сутегі қоспасының толық жануы, азот оксидтерінің аз мөлшерде түзілуі және күйе бөлінуінің төмендеуі үшін сутегінің көлемдік концентрациясы 70%-ға дейін жетуі мүмкін екені көрсетілген. Сонымен бірге метанға 50% сутегі қосу жану жылдамдығы мен жану шектеріне айтарлықтай әсер етпейтіні де байқалған [48]. Зерттеу авторлары қоспадағы сутегі концентрациясының артуы жоғары температура аймағында жану кедергісін едәуір төмендететінін атап көрсетеді. Бұл құбылыс температура жоғарылаған сайын сутегі атомдарының қатысуымен жүретін реакциялардың қарқындылығының артуымен түсіндіріледі.

H2	1	38.000	2.920	0.000	0.790	280.000
HE	0	10.200	2.576	0.000	0.000	0.000
C6H6	2	468.500	5.230	0.000	10.30	1.000
H2O	2	572.400	2.605	1.844	0.000	4.000

↑

Name of the species
(must be the same
reported in the
thermodynamic and
kinetic files)

↑

The potential well
depth (in K).

↑

The collision diameter
(in angstroms)

↑

The dipole moment
(in Debye)

↑

The polarizability (in
cubic angstroms)

↑

The rotational relaxation
collision number at 298 K

↑

An index indicating the geometrical
configuration. If the index is 0, the
molecule is a single atom. If the index is
1 it is linear, and if it is 2, the molecule
is nonlinear.

Сурет 10 – Кинетикалық механизмнің көліктік мәліметтері

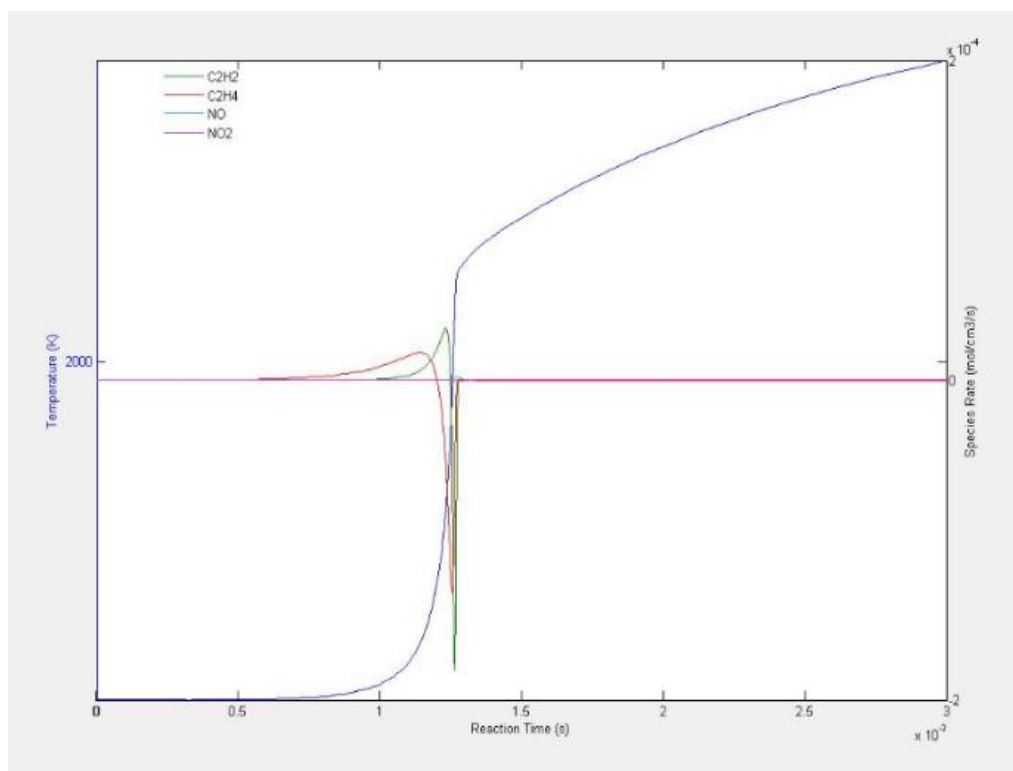
309 реакциядан және 53 реагенттен тұратын GRI-Mech кинетикалық механизмі бағдарламаның деректер қорынан тандап алынды. Ауа ортасында метанның жану үдерісін модельдеу PrIme бағдарламалық кешенінің көмегімен орындалды. Жану үдерісі Plug Flow Reactor типті реакторда жүргізілді.

Қоспаның стехиометриялық құрамы 1 моль = 100% деп қабылданған жағдайда келесі түрде берілді: $\text{CH}_4 = 3.33\%$, $\text{O}_2 = 6.66\%$, $\text{N}_2 = 90\%$. Бастапқы температура $T = 1688 \text{ K}$, қысым $p = 0.39 \text{ bar}$, ал реактордағы болу уақыты $t = 3 \text{ s}$ деп алынды.

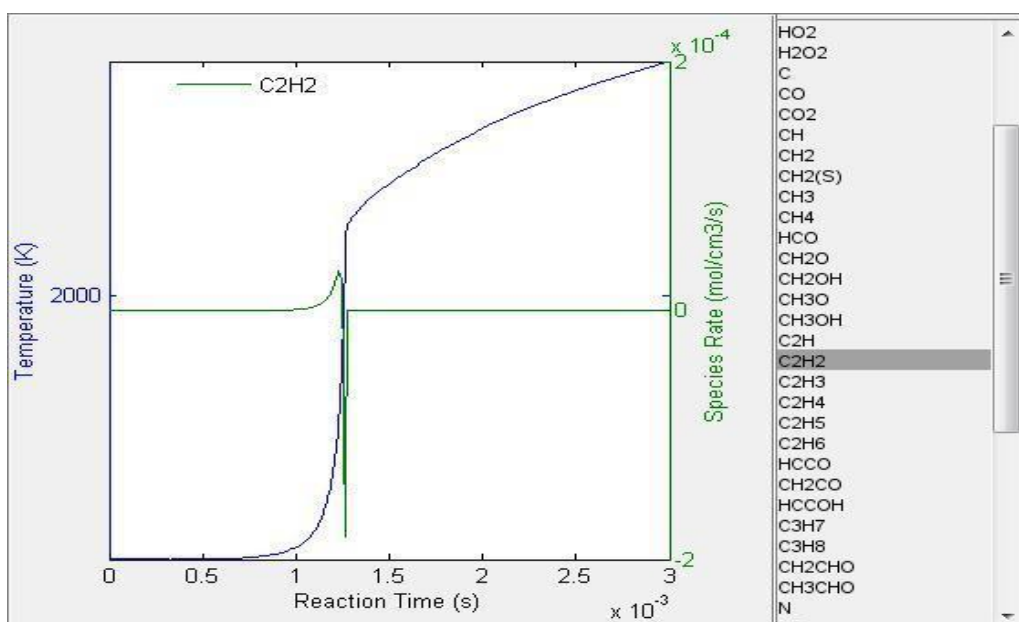
Модельдеу нәтижесінде тура және кері реакциялардың жылдамдықтары анықталды. Сонымен қатар әрбір реагенттің түзілу жылдамдығы мен концентрациясының өзгерісін бақылауға мүмкіндік алынды. 11, 12 және 13-суреттерде көмірсутектердің түзілуі, сондай-ақ температураға байланысты метанның жұмсалыу үдерісінің модельдеу нәтижелері көрсетілген.

Реактивті отындардың жануы кезінде және жалпы жағдайда құрамында көміртегі бар заттардың химиялық түрленуі барысында күйенің, яғни дисперсті көміртегінің түзілу механизмі әлі де толық зерттелмеген күрделі үдерістердің бірі болып табылады. Бұл бағыттағы зерттеулерде күйе түзілуінде полимерлену үдерістері немесе тармақталған тізбекті реакциялар маңызды рөл атқаруы мүмкін екені қарастырылады.

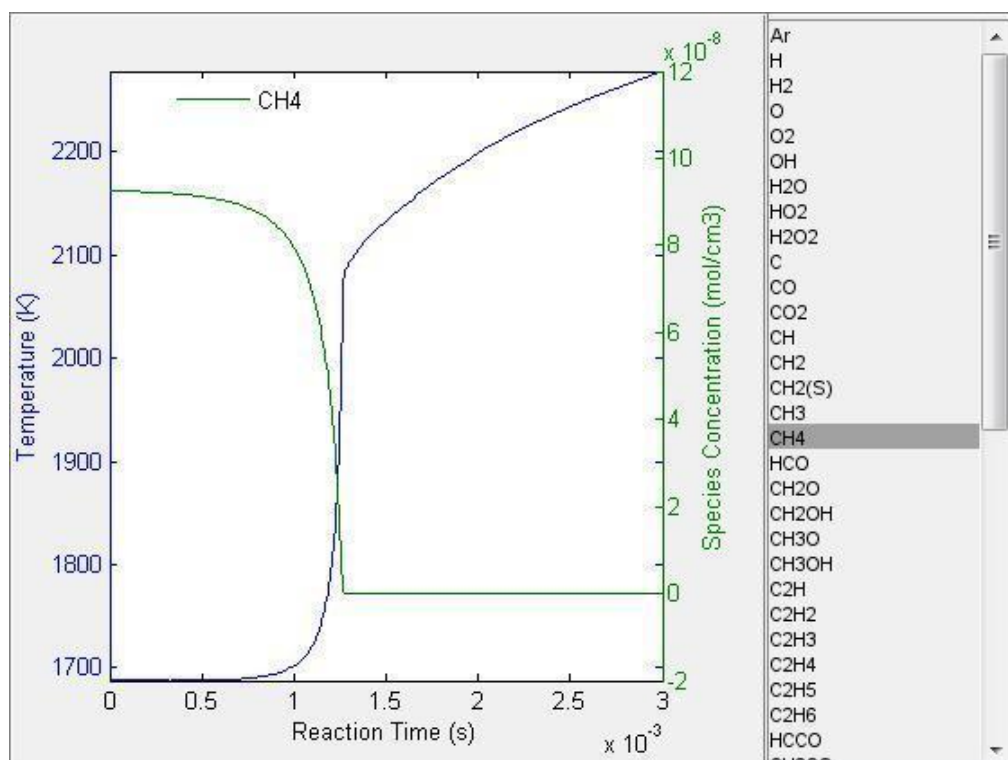
Пеш ішіндегі газ ағындарының аэродинамикалық ерекшеліктеріне байланысты күйе бөлшектері алау аймағында немесе одан тыс кеңістікте орналасуы мүмкін. Егер күйе бөлшектері алаудан бөлініп, температурасы 800 °C-тан төмен аймаққа өтсе, олар толық жанып үлгермей, жану камерасынан сыртқа шығарылады. Ал алау ішінде қалған күйе бөлшектері экрандық қыздыру беттерінің әсерінен жану өнімдерінің температурасы төмендегенге дейін тотығу және жану үдерісін жалғастырады.



Сурет 11 – Ластағыш заттардың пайда болуы



Сурет 12 – Ацетилен концентрациясы



Сурет 13 – Метанның концентрациясы

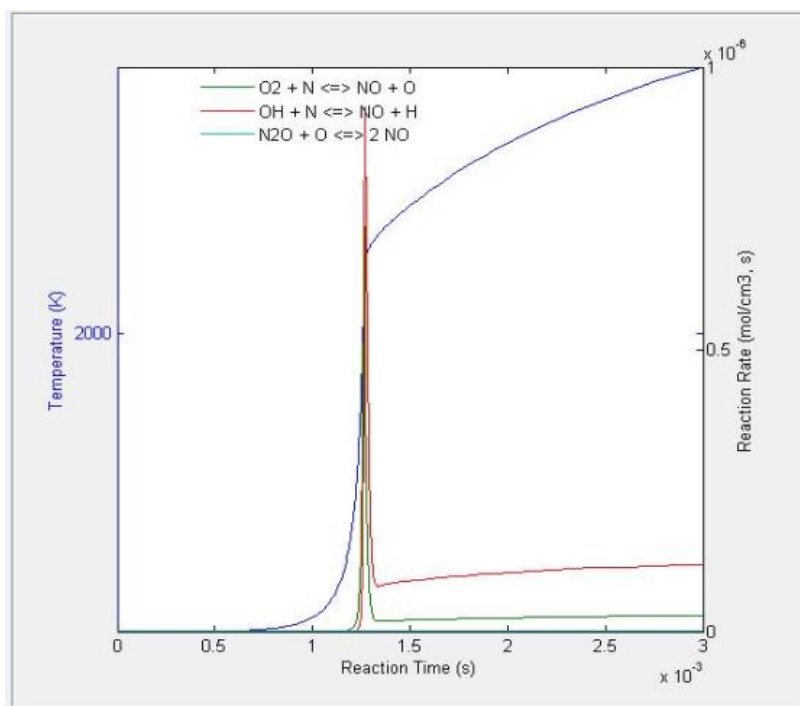
Сонымен қатар жану үдерісі барысында NO_x сияқты басқа да ластаушы заттар түзіледі. NO_x түзілуі оттегінің жұмсалу жылдамдығына және жану кезіндегі өзара бәсекелес химиялық реакциялардың жүру қарқындылығына тәуелді болады. Отынның пиролизі нәтижесінде түзілетін атомдар мен радикалдар ауа құрамындағы азотпен әрекеттесіп, соңында NO_x қосылыстарының пайда болуына әкеледі.

Мұндай түрленулердің жылдамдығы бірнеше факторлармен анықталады.

Оларға реакция аймағындағы азот концентрациясы, табиғи газ-ауа қоспасының жану жылдамдығы, сондай-ақ белсенді орталықтар үшін жүретін бәсекелес реакциялардың жылдамдығы жатады.

Ауа құрамында азот мөлшері шектеулі болған жағдайда және температура 1500 К-ден төмен болған кезде молекулалық азот тек O_2 молекулаларымен соқтығысу арқылы ғана емес, сонымен қатар CH_4 молекуласының пиролизі нәтижесінде пайда болатын көмірсутекті фрагменттермен де әрекеттеседі. Бұл жағдайда HCN және CN типті қосылыстар NO түзілуінің тізбекті механизміне қатысатын аралық өнімдер ретінде қарастырылады.

Дегенмен жану кезінде азот оксидтерінің түзілу шарттары әлі де толық зерттелмеген. Бұл үдеріс күрделі химиялық кинетикамен, жылу және масса алмасу құбылыстарымен, сондай-ақ олардың өзара әсерімен тығыз байланысты. Сондықтан NO_x түзілу механизмін нақты сипаттау үшін химиялық реакциялар кинетикасын жылу және масса тасымалдау үдерістерімен бірге кешенді түрде зерттеу қажет. 14-суретте NO_x азот оксидтерінің түзілу үдерісі көрсетілген.



Сурет 14 – NO_x қалыптасуы

Қазіргі уақытта метанның жануын сипаттауға арналған механизм NO_x түзілу үдерісін ескере отырып тексерілуде және Chemical Workbench 4.0 бағдарламасында тәжірибелік деректермен салыстырылуда. Ацетиленнің жануы мен азот оксидтерінің түзілуін сипаттау үшін, сондай-ақ кейбір реакциялардың көмірсутектердің түзілуіне әсерін нақтылау мақсатында GRI-Mech механизміне қосымша реакциялар енгізілді.

Кейбір шектеулеріне қарамастан, GRI-Mech механизмі метанның, яғни табиғи газдың жану үдерісін жеткілікті дәлдікпен сипаттайтындықтан негізгі механизм ретінде таңдап алынды. Жаңа механизмді құрастыру барысында қосылған реакциялар AramcoMech және n-Butane сияқты ірі кинетикалық

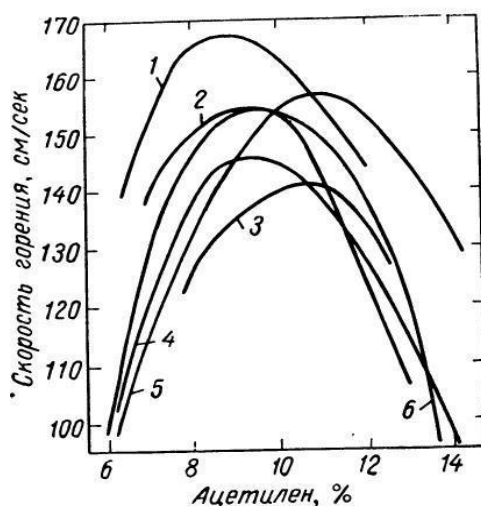
механизмдерден алынды. Бұл механизмдер үлкен көмірсутектердің жану үдерістерін сипаттауға арналған.

Көмірсутектердің жануына арналған кинетикалық механизмдерді генерациялау алгоритмін қолдану арқылы метан механизмі негізінде ацетиленнің жануын модельдеуге арналған жаңа кинетикалық механизм құрастырылды. Бұл механизм полиароматикалық көмірсутектер мен күйе бөлшектерінің түзілуін әрі қарай зерттеу үшін қолданылды. Құрастырылған механизм 380 реакциядан және 84 реагенттен тұрады және берілген жағдайларда ацетиленнің жану үдерісін сипаттайды.

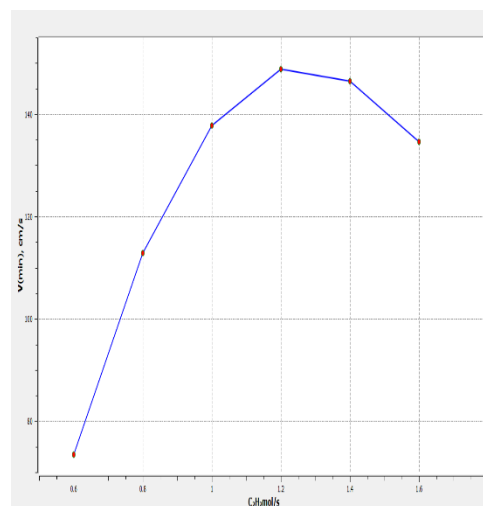
Модельдеу нәтижелерін тексеру үшін алынған есептеу деректері 2006 жылы Ю. Варнатц жетекшілігімен орындалған Crina I. Heghes, Chem. Eng. “C₁–C₄ Hydrocarbon Oxidation Mechanism”, Heidelberg диссертациялық жұмысындағы тәжірибелік мәліметтермен салыстырылды.

Модельдеу үшін бастапқы параметрлер келесідей алынды: T = 298 K, p = 1 bar. Қоспа құрамындағы ацетилен концентрациясы C₂H₂ = 0.6–1.8 моль, оттегі концентрациясы O₂ = 2.5 моль, ал азот концентрациясы N₂ = 9.4 моль деп берілді. Ацетиленнің жану жылдамдығы 15-суретте көрсетілген.

15-суретте келтірілген нәтижелер тәжірибелік деректермен толық сәйкес келмейді. Мұндай айырмашылықтар тәжірибелердің әртүрлі құрылғыларда жүргізілуімен және олардың орындалу шарттарының бірдей болмауымен түсіндіріледі. Сонымен қатар кейбір тәжірибелік зерттеулердің бұрынғы кезеңдерде орындалуы да нәтижелер арасындағы алшақтыққа әсер етуі мүмкін.



а



б

а – 1, 2 - құбыр; 3 – жетілген құбыр, 4 – бунзеннің қаныққан буымен жалыны; 5 – сабынды көбіктер; 6 – бунзендік жалын, б – CWB бағдарламасы

Сурет 15 – Әр түрлі жағдайлармен ауада ацетиленнің жануын моделдеу

Тәжірибелік зерттеулер үшін ауа мен ацетиленнің стехиометриялық қоспасы таңдап алынды. Мұндай қоспаны қолданудың себебі — стехиометриялық жағдайда күйе тәрізді өнімдер мен NO азот оксидінің салыстырмалы түрде аз мөлшерде түзілуі. Ацетиленнің элементтік құрамы $C_2 = 92.3\%$ және $H_2 = 7.7\%$ түрінде сипатталады. Бұл ацетиленнің көміртекке бай қосылыс екенін және жану барысында күйе мен басқа да көміртекті бөлшектердің түзілуіне бейім екенін көрсетеді.

Қоспада ацетилен мөлшерінің артуы бос көміртектің көбеюіне әкеледі. Соның нәтижесінде жалынның ішкі ядросы ұлғайып, қызған күйе бөлшектерімен қаныққан жарық шығару аймағы кеңейеді. Ацетилен концентрациясы одан әрі артқан жағдайда пирогендік ыдыраудың соңғы өнімдері — CO және H_2 түзілетін жалынның ортаңғы аймағы, яғни тұтану аймағы, біртіндеп жойылып, жалын қатты түтіндене бастайды.

Тотықсыздану сатысы жалынның ортаңғы аймағында жүретін үдерістермен сипатталады. Бұл аймақта тотығу реакциялары күшейіп, CO және H_2 қосылыстарының сәйкесінше көмірқышқыл газына және су буына дейін белсенді тотығуы басталады: $CO \rightarrow CO_2$ және $H_2 \rightarrow H_2O$.

Жанудан кейінгі кезеңде жалын алауында ацетиленнің пирогендік ыдырау өнімдерінің қарқынды тотығуы, яғни жануы жүреді. Бұл үдерістің нәтижесінде соңғы жану өнімдері ретінде CO_2 және H_2O түзіледі.

3.2 CUDA негізіндегі кинетикалық солвер алгоритмінің құрылымы

Жану процестерін модельдеу кезінде химиялық реакциялар жүйесін сипаттайтын дифференциалдық теңдеулерді сандық түрде шешу қажет болады. Мұндай теңдеулер жүйесі көптеген химиялық компоненттер мен реакциялардан тұрады. Реакциялар саны артқан сайын есептеу күрделілігі де өседі. Сондықтан үлкен реакциялар жүйесін есептеу үшін жоғары өнімді есептеу әдістерін қолдану қажет. Осы жұмыста химиялық кинетика теңдеулерін шешу үшін графикалық процессорларды пайдаланатын CUDA негізіндегі кинетикалық солвер алгоритмі ұсынылады. Химиялық кинетика теңдеулерін GPU архитектурасында шешу соңғы жылдары жану процестерін жоғары өнімді модельдеудің маңызды бағытына айналды [49–51].

CUDA технологиясы графикалық процессорларды жалпы мақсаттағы ғылыми есептеулерде қолдануға мүмкіндік береді. GPU құрылғысы көптеген есептеу ядроларынан тұрады және олар бір уақытта мыңдаған есептеу ағындарын орындай алады. Мұндай архитектура біртекті математикалық операцияларды үлкен көлемде параллель орындауға мүмкіндік береді. Химиялық кинетика теңдеулерін есептеу кезінде реакциялар жүйесінде ұқсас есептеу операциялары көп рет қайталанатындықтан, бұл есеп GPU архитектурасы үшін тиімді болып табылады.

Ұсынылған кинетикалық солвер алгоритмінің негізгі мақсаты – химиялық реакциялар жүйесін сипаттайтын теңдеулерді уақыт бойынша интегралдау және химиялық компоненттердің концентрацияларының өзгерісін анықтау. Бұл есептеулерді орындау үшін Runge–Kutta сандық әдісі қолданылады.

Алгоритмнің псевдокоды есептеу процесінің негізгі кезеңдерін сипаттайды және GPU негізіндегі есептеулердің жалпы логикасын көрсетеді.

Алгоритмнің кіріс параметрлері химиялық жүйенің сипаттамаларын анықтайды. Оларға реакциялар саны, компоненттер саны, уақыт қадамы және интегралдау қадамдарының саны жатады. Сонымен қатар алгоритмге реакция константалары мен стехиометриялық коэффициенттер массивтері беріледі. Бұл параметрлер химиялық реакциялар жүйесінің кинетикалық қасиеттерін анықтайды.

Алгоритмнің бірінші кезеңінде CPU жадысында негізгі деректер массивтері инициализацияланады. Бұл массивтер химиялық компоненттердің концентрацияларын, реакция константаларын және стехиометриялық коэффициенттерді сақтау үшін қолданылады. Стехиометриялық коэффициенттер екі өлшемді матрицалар түрінде сақталады. Бірінші матрица реагенттердің коэффициенттерін сипаттайды, ал екінші матрица реакция өнімдерінің коэффициенттерін сипаттайды.

CPU жадысында деректер инициализацияланғаннан кейін GPU жадысында сәйкес массивтер бөлінеді. GPU жадысында есептеу кезінде қолданылатын барлық негізгі параметрлер сақталады. Сонымен қатар Runge–Kutta әдісінің аралық коэффициенттерін сақтау үшін қосымша массивтер қолданылады. Бұл коэффициенттер k_1 , k_2 , k_3 , k_4 және k_5 массивтерінде сақталады. Бұл массивтер уақыт бойынша интегралдау кезінде концентрациялардың өзгерісін есептеу үшін пайдаланылады.

Келесі кезеңде барлық бастапқы деректер CPU жадысынан GPU жадысына көшіріледі. Бұл операция CUDA кітапханасының **cudaMemcpy()** функциясы арқылы жүзеге асырылады. Бұл функция хост пен құрылғы арасындағы деректер алмасуды қамтамасыз етеді. GPU жадысына реакция константалары, стехиометриялық коэффициенттер және бастапқы концентрациялар көшіріледі.

Деректер GPU жадысына көшірілгеннен кейін алгоритмнің негізгі есептеу кезеңі басталады. Бұл кезең уақыт бойынша интегралдау циклін қамтиды. Әрбір уақыт қадамында химиялық жүйенің жаңа күйі есептеледі. Бұл есептеу бірнеше кезеңнен тұратын Runge–Kutta әдісі арқылы жүзеге асырылады.

Runge–Kutta әдісі дифференциалдық теңдеулер жүйесін жоғары дәлдікпен интегралдауға мүмкіндік береді. Бұл әдіс бірнеше аралық коэффициенттерді есептеу арқылы келесі уақыт қадамындағы мәнді анықтайды. Әрбір уақыт қадамында Runge–Kutta әдісінің бірнеше кезеңі орындалады. Әрбір кезеңде реакция жылдамдығы есептеледі және концентрациялардың өзгерісі анықталады.

Химиялық реакция жылдамдығы әрекет етуші массалар заңына сәйкес анықталады. Реакция жылдамдығы реагенттердің концентрациялары мен реакция константаларына тәуелді болады.

Реакция жылдамдығы есептелгеннен кейін химиялық компоненттердің концентрацияларының өзгерісі анықталады. Бұл өзгеріс реакциядағы реагенттер мен өнімдердің стехиометриялық коэффициенттеріне байланысты болады.

Алынған концентрация өзгерістері Runge–Kutta әдісінің аралық коэффициенттерін есептеу үшін қолданылады. Әрбір уақыт қадамында k_1 , k_2 , k_3 , k_4 және k_5 коэффициенттері есептеледі. Бұл коэффициенттер концентрацияның келесі уақыт қадамындағы мәнін анықтау үшін пайдаланылады.

Алгоритмнің негізгі есептеу кезеңдері GPU kernel функциялары арқылы жүзеге асырылады. Kernel іске қосылған кезде көптеген CUDA ағындары бір уақытта жұмыс істейді. Әрбір CUDA ағыны жеке реакция үшін есептеу жүргізеді. Бұл тәсіл барлық реакцияларды бір уақытта өңдеуге мүмкіндік береді.

Есептеу процесі уақыт бойынша цикл арқылы ұйымдастырылады. Әрбір уақыт қадамында Runge–Kutta әдісінің бірнеше кезеңі орындалады. Әрбір кезеңде GPU kernel функциялары іске қосылады және реакциялар жүйесі параллель түрде өңделеді.

Есептеу аяқталғаннан кейін алынған концентрациялар GPU жадысынан CPU жадысына көшіріледі. Бұл операция **cudaMemcpy()** функциясы арқылы жүзеге асырылады. CPU жадысына көшірілген нәтижелер файлға жазылады және кейінгі талдау үшін сақталады.

Алгоритмнің соңғы кезеңінде GPU жадысында бөлінген барлық массивтер босатылады. Бұл операция **cudaFree()** функциясы арқылы орындалады. GPU жадысын дұрыс босату бағдарламаның тұрақты жұмыс істеуін қамтамасыз етеді.

Осылайша CUDA негізіндегі кинетикалық солвер алгоритмінің псевдокоды химиялық реакциялар жүйесін параллель түрде есептеудің жалпы құрылымын сипаттайды. GPU архитектурасының параллель мүмкіндіктерін пайдалану нәтижесінде көптеген реакциялар бір уақытта өңделеді және есептеу уақыты айтарлықтай қысқарады. Бұл тәсіл күрделі жану процестерін модельдеу кезінде жоғары тиімділікті қамтамасыз етеді. CUDA негізіндегі кинетикалық солверде реакциялар жүйесін параллель өңдеу есептеу уақытын қысқартуға және үлкен механизмдерді тиімді есептеуге мүмкіндік береді [52–54].

3.2.1 GPU kernel ішіндегі есептеу барысы

CUDA технологиясында негізгі есептеу операциялары графикалық процессорда орындалатын kernel функциялары арқылы жүзеге асырылады. Kernel – бұл GPU құрылғысында параллель түрде орындалатын арнайы функция. Kernel іске қосылған кезде көптеген CUDA ағындары (threads) бір уақытта жұмыс істейді және әрбір ағын деректердің белгілі бір бөлігін өңдейді. Мұндай тәсіл біртепті есептеу операцияларын үлкен көлемде параллель орындауға мүмкіндік береді.

Ұсынылған кинетикалық солверде химиялық реакциялар жүйесін сипаттайтын негізгі есептеулер GPU kernel функциялары ішінде орындалады. Kernel функциясының әрбір ағыны жеке химиялық реакция үшін есептеу жүргізеді. Бұл тәсіл реакциялар жүйесін толық параллель түрде өңдеуге мүмкіндік береді. Осылайша әрбір реакцияның жылдамдығы жеке CUDA ағынында есептеледі.

kernel іске қосылған кезде көптеген CUDA ағындары бір уақытта жұмыс істейді. Әрбір ағын белгілі бір реакция индексін алады және сол реакция үшін

барлық есептеулерді орындайды. Бұл тәсіл реакциялар жүйесін толық параллель түрде есептеуге мүмкіндік береді. GPU kernel ішінде әрбір ағынның жеке реакцияны өңдеуі химиялық source term есептеулерін параллель орындауға мүмкіндік береді [55, 56].

Алгоритмнің бірінші кезеңінде әрбір ағын өзінің өңдейтін реакция индексін анықтайды. Бұл индекс арқылы реакция параметрлері массивтерінен қажетті деректер алынады. Әрбір реакция үшін реагенттер мен өнімдердің концентрацияларының көбейтінділері есептеледі. Бұл шамалар реакция жылдамдығын анықтау үшін қолданылады.

Реагенттердің концентрация көбейтіндісі анықталады. Ұқсас түрде реакция өнімдерінің концентрацияларының көбейтіндісі есептеледі. Бұл көбейтінділер реакция жылдамдығын есептеу үшін қолданылады. Химиялық кинетика теориясына сәйкес реакция жылдамдығы әрекет етуші массалар заңына негізделеді. Реакция жылдамдығы келесі түрде анықталады.

Реакция жылдамдығы анықталғаннан кейін химиялық компоненттердің концентрация өзгерісі есептеледі. Бұл өзгеріс реакциядағы реагенттер мен өнімдердің стехиометриялық коэффициенттері арқылы анықталады.

Алынған концентрация өзгерістері Runge–Kutta әдісінің аралық коэффициенттерін есептеу үшін қолданылады. Kernel ішінде есептелген мәндер k_1 , k_2 , k_3 , k_4 және k_5 массивтеріне қосылады. Бұл массивтер кейін концентрацияларды жаңарту кезінде пайдаланылады.

GPU архитектурасында көптеген CUDA ағындары бір уақытта жұмыс істейтіндіктен, бірнеше ағын бір жады ұяшығына жазуға тырысуы мүмкін. Мұндай жағдайда синхронизация мәселесі туындайды. Егер бірнеше ағын бір уақытта бір айнымалының мәнін өзгертуге тырысса, онда есептеу нәтижесі дұрыс болмауы мүмкін. Бұл жағдай race condition деп аталады.

Ұсынылған алгоритмде бұл мәселені шешу үшін атомарлық операциялар қолданылады. Атомарлық операциялар бірнеше ағын бір уақытта бір жады ұяшығына қол жеткізген жағдайда операцияның дұрыс орындалуын қамтамасыз етеді. Алгоритмде концентрация өзгерістерін жинақтау үшін **atomicAdd()** функциясы қолданылады.

atomicAdd () функциясы бірнеше CUDA ағындары бір уақытта бір айнымалыға мән қосқан жағдайда операцияның толық орындалуын қамтамасыз етеді. Бұл операция орындалған кезде басқа ағындар осы жады ұяшығына қол жеткізуді уақытша күтіп тұрады. Осылайша деректердің бұзылуының алдын алуға болады. Бірнеше ағын бір компоненттің концентрация өзгерісіне үлес қосатындықтан, нәтижелерді дұрыс жинақтау үшін атомарлық операциялар немесе редукция әдістері қолданылады [57, 58].

GPU kernel ішінде орындалатын есептеу алгоритмі химиялық реакциялар жүйесін толық параллель түрде өңдеуге мүмкіндік береді. Әрбір реакция жеке CUDA ағынында есептелетіндіктен, көптеген реакциялар бір уақытта өңделеді. Бұл тәсіл үлкен реакциялар жүйесін модельдеу кезінде есептеу уақытын айтарлықтай қысқартуға мүмкіндік береді.

Осылайша GPU kernel ішіндегі есептеу псевдокоды химиялық кинетика теңдеулерін GPU архитектурасында параллель түрде есептеудің негізгі принциптерін сипаттайды. Kernel функциялары арқылы орындалатын параллель есептеулер Runge–Kutta әдісінің әрбір кезеңін жылдам орындауға мүмкіндік береді және жалпы есептеу тиімділігін арттырады.

3.2.2 Бағдарламаның жалпы алгоритмі

CUDA негізіндегі кинетикалық солвер бағдарламасының жалпы алгоритмі химиялық реакциялар жүйесін уақыт бойынша сандық интегралдау үшін арналған. Бағдарлама химиялық кинетика теңдеулерін шешу кезінде CPU және GPU есептеу ресурстарын біріктіріп қолданады. CPU бағдарламаның жалпы басқаруын жүзеге асырады, ал негізгі математикалық есептеулер GPU құрылғысында параллель түрде орындалады. Мұндай тәсіл үлкен реакциялар жүйесін тиімді түрде есептеуге мүмкіндік береді.

Бағдарламаның жалпы алгоритмі бірнеше негізгі кезеңнен тұрады. Бұл кезеңдерге бастапқы деректерді оқу, жадыны инициализациялау, GPU жадысында есептеу құрылымдарын дайындау, уақыт бойынша интегралдау, нәтижелерді сақтау және есептеу ресурстарын босату жатады. Бағдарламаның барлық жұмыс логикасы осы кезеңдердің ретімен орындалуына негізделеді.

Алгоритмнің бірінші кезеңінде бағдарламаның бастапқы параметрлері анықталады. Бұл параметрлер химиялық жүйенің негізгі сипаттамаларын қамтиды. Оларға реакциялар саны, химиялық компоненттер саны, уақыт қадамы және уақыт бойынша интегралдау қадамдарының саны жатады. Сонымен қатар реакция константалары мен стехиометриялық коэффициенттер массивтері инициализацияланады. Бұл параметрлер кейінгі есептеулерде қолданылады.

Келесі кезеңде CPU жадысында негізгі деректер құрылымдары дайындалады. Бұл құрылымдар химиялық реакциялар жүйесінің параметрлерін сақтау үшін қолданылады. Бағдарламада бірнеше негізгі массивтер қолданылады. Оларға химиялық компоненттердің концентрациялары, реакция константалары және стехиометриялық коэффициенттер жатады. Сонымен қатар уақыт бойынша есептеу нәтижелерін сақтау үшін қосымша массивтер қолданылады.

CPU жадысында деректер дайындалғаннан кейін GPU жадысында сәйкес массивтер бөлінеді. GPU жадысында сақталатын массивтер есептеу кезінде қолданылатын барлық негізгі параметрлерді қамтиды. Бұл массивтер кейін GPU kernel функцияларында қолданылады. GPU жадысын дұрыс ұйымдастыру есептеу тиімділігіне тікелей әсер етеді, себебі деректерге қол жеткізу жылдамдығы есептеу өнімділігіне маңызды әсер етеді.

Бағдарламаның келесі кезеңінде барлық бастапқы деректер CPU жадысынан GPU жадысына көшіріледі. Бұл операция CUDA кітапханасының **cudaMemcpy()** функциясы арқылы жүзеге асырылады. Бұл функция хост пен құрылғы арасындағы деректер алмасуды қамтамасыз етеді. GPU жадысына реакция параметрлері, концентрациялар және стехиометриялық коэффициенттер көшіріледі. Осыдан кейін GPU құрылғысы есептеуге дайын күйге келеді.

Алгоритмнің негізгі бөлігі уақыт бойынша интегралдау циклінен тұрады. Бұл цикл әрбір уақыт қадамында химиялық жүйенің жаңа күйін есептейді. Уақыт бойынша интегралдау дифференциалдық теңдеулер жүйесін шешу арқылы жүзеге асырылады. Бұл есептеу Runge–Kutta сандық әдісінің көмегімен орындалады.

Уақыт бойынша интегралдау кезінде әрбір уақыт қадамында бірнеше есептеу операциялары орындалады. Алдымен химиялық реакциялар жылдамдықтары анықталады. Реакция жылдамдығы әрекет етуші массалар заңына сәйкес есептеледі. Реакция жылдамдығы реагенттердің концентрациялары мен реакция константаларына тәуелді болады.

$$r_i = k_r[i] \cdot mult2 - k_f[i] \cdot mult1 \quad (28)$$

мұндағы, r_i — i -ші реакцияның жылдамдығы, k_f және k_r сәйкесінше тура және кері реакция константалары болып табылады.

Реакция жылдамдықтары анықталғаннан кейін химиялық компоненттердің концентрацияларының өзгерісі есептеледі. Бұл өзгеріс барлық реакциялар әсерін ескере отырып анықталады.

$$\Delta C_k = \sum_{i=1}^N (left_{i,k} - right_{i,k}) r_i \quad (29)$$

мұндағы ΔC_k — k -ші химиялық компоненттің концентрациясының өзгерісі.

Runge–Kutta әдісі дифференциалдық теңдеулер жүйесін уақыт бойынша интегралдау үшін қолданылады. Бұл әдіс бірнеше аралық коэффициенттерді пайдаланып келесі уақыт қадамындағы концентрация мәнін анықтайды.

$$C_{n+1} = C_n + \frac{dt}{6} (k_1 + 2k_2 + 2k_3 + k_4) \quad (30)$$

мұндағы C_n — ағымдағы уақыттағы концентрация, C_{n+1} — келесі уақыт қадамындағы концентрация.

Уақыт бойынша интегралдау кезінде барлық негізгі есептеулер GPU kernel функциялары арқылы орындалады. Kernel іске қосылған кезде көптеген CUDA ағындары бір уақытта жұмыс істейді. Әрбір CUDA ағыны жеке реакция үшін есептеу жүргізеді. Бұл тәсіл барлық реакциялар жүйесін толық параллель түрде өңдеуге мүмкіндік береді.

Бағдарлама бірнеше негізгі кезеңнен тұрады. Алдымен барлық бастапқы параметрлер анықталады және деректер құрылымдары инициализацияланады. Содан кейін GPU жадысында есептеу үшін қажетті массивтер бөлінеді және барлық деректер GPU жадысына көшіріледі.

Осыдан кейін негізгі есептеу кезеңі басталады. Уақыт бойынша цикл ішінде GPU kernel функциялары іске қосылады және химиялық реакциялар

жүйесі параллель түрде өңделеді. Әрбір уақыт қадамында концентрациялар жаңартылады және жаңа мәндер есептеледі.

Есептеу аяқталғаннан кейін алынған нәтижелер GPU жадысынан CPU жадысына көшіріледі. Бұл нәтижелер файлға жазылады және кейінгі талдау үшін қолданылады.

Бағдарламаның соңғы кезеңінде GPU жадысында бөлінген барлық ресурстар босатылады. Бұл операция **cudaFree()** функциясы арқылы орындалады. GPU жадысын дұрыс босату бағдарламаның тұрақты жұмыс істеуін қамтамасыз етеді. Host және device арасындағы деректер алмасу, GPU жадысын бөлу және есептеу ядроларын іске қосу CUDA бағдарламаларының негізгі кезеңдеріне жатады [59, 60].

Осылайша бағдарламаның жалпы алгоритмі химиялық кинетика теңдеулерін GPU архитектурасында тиімді түрде есептеуге мүмкіндік береді. CPU және GPU ресурстарын біріктіріп қолдану нәтижесінде үлкен реакциялар жүйесін есептеу уақыты айтарлықтай қысқарады және есептеу тиімділігі артады.

3.2.3 Бағдарламаның деректер құрылымы

CUDA негізіндегі кинетикалық солвер бағдарламасында деректер құрылымын дұрыс ұйымдастыру есептеу тиімділігінің маңызды элементтерінің бірі болып табылады. Химиялық кинетика теңдеулерін шешу кезінде көптеген параметрлер мен айнымалылар қолданылады. Оларға химиялық компоненттердің концентрациялары, реакция константалары, стехиометриялық коэффициенттер және аралық есептеу нәтижелері жатады. Бұл деректердің барлығы арнайы массивтер мен құрылымдарда сақталады және есептеу процесінде белсенді түрде пайдаланылады.

Бағдарламаның деректер құрылымы химиялық реакциялар жүйесінің параметрлерін тиімді сақтауға және GPU архитектурасында параллель есептеулерді орындауға мүмкіндік беретіндей ұйымдастырылған. Деректерді ұйымдастыру кезінде олардың өлшемдері, орналасуы және есептеу кезінде қолданылу ерекшеліктері ескеріледі. Бұл деректерге жылдам қол жеткізуге және есептеу операцияларын тиімді орындауға мүмкіндік береді.

Бағдарламада қолданылатын негізгі параметрлерге реакциялар саны N және химиялық компоненттер саны M жатады. Реакциялар саны химиялық жүйеде қанша реакция бар екенін анықтайды, ал компоненттер саны жүйеде қанша химиялық зат қатысатынын көрсетеді. Осы параметрлерге байланысты барлық негізгі массивтердің өлшемдері анықталады.

Бағдарламада химиялық компоненттердің концентрацияларын сақтау үшін бір өлшемді массив қолданылады. Бұл массивте әрбір элемент белгілі бір химиялық компоненттің концентрациясын сипаттайды.

Химиялық реакциялардың кинетикалық параметрлерін сипаттау үшін реакция константаларының массивтері қолданылады. Реакция константалары екі түрге бөлінеді: тура реакция константалары және кері реакция константалары. Бұл константалар келесі массивтерде сақталады:

- k_f — тура реакция константалары
- k_r — кері реакция константалары

Бұл массивтердегі элементтер реакциялар санына тең болады.

$$k_f = \{k_{f1}, k_{f2}, \dots, k_{fN}\} \quad (31)$$

$$k_r = \{k_{r1}, k_{r2}, \dots, k_{rN}\} \quad (32)$$

мұндағы k_{fi} және k_{ri} сәйкесінше i -ші реакцияның тура және кері реакция константалары болып табылады.

Химиялық реакциялардың құрамын сипаттау үшін стехиометриялық коэффициенттер матрицалары қолданылады. Бұл коэффициенттер реагенттер мен реакция өнімдерінің құрамын сипаттайды. Бағдарламада екі негізгі матрица қолданылады:

- **left** — реагенттердің стехиометриялық коэффициенттері
- **right** — өнімдердің стехиометриялық коэффициенттері

Бұл матрицалардың өлшемі $N \times MN \times MN \times M$ болады, яғни әрбір реакция үшін барлық химиялық компоненттердің коэффициенттері көрсетіледі.

$$L = [l_{ij}], \quad i = 1..N, j = 1..M \quad (33)$$

$$R = [r_{ij}], \quad i = 1..N, j = 1..M \quad (34)$$

мұндағы, l_{ij} — i -ші реакциядағы j -ші компоненттің реагент коэффициенті, ал r_{ij} — сол компоненттің реакция өніміндегі коэффициенті.

Runge–Kutta әдісін қолдану кезінде концентрациялардың өзгерісін есептеу үшін аралық коэффициенттер қолданылады. Бұл коэффициенттер есептеу процесінде уақытша мәндерді сақтау үшін қажет. Бағдарламада келесі массивтер қолданылады: k_1, k_2, k_3, k_4, k_5

Бұл массивтердің өлшемі химиялық компоненттер санына тең болады. Runge–Kutta әдісінің әрбір кезеңінде осы массивтердің элементтері есептеледі және кейін концентрацияларды жаңарту кезінде қолданылады.

Runge–Kutta коэффициенттері концентрацияның уақыт бойынша өзгерісін есептеу үшін қолданылады. Әрбір коэффициент дифференциалдық тендеулер жүйесінің белгілі бір уақыт нүктесіндегі мәнін сипаттайды.

Бағдарламада есептеу нәтижелерін уақытша сақтау үшін қосымша массивтер қолданылады. Мысалы, жаңа концентрация мәндерін сақтау үшін арнайы массив пайдаланылады. Бұл массив уақыт бойынша интегралдау кезінде аралық нәтижелерді сақтауға мүмкіндік береді.

Бағдарламаның деректер құрылымы GPU архитектурасында параллель есептеулерді орындауға бейімделген. Барлық негізгі деректер массивтер түрінде ұйымдастырылған. Мұндай құрылым CUDA kernel функцияларында деректерге тиімді қол жеткізуге мүмкіндік береді. Сонымен қатар массивтерді қолдану GPU жадысын тиімді пайдалануға мүмкіндік береді.

Деректер құрылымын ұйымдастыру кезінде тағы бір маңызды фактор – деректердің өлшемдерін дұрыс анықтау. Массивтердің өлшемдері реакциялар саны мен компоненттер санына байланысты болады. Сондықтан бағдарламаны іске қосу алдында осы параметрлер нақты анықталуы қажет.

Бағдарламаның деректер құрылымы бірнеше негізгі массивтерден тұрады. Бұл массивтер химиялық реакциялар жүйесінің барлық параметрлерін сақтауға арналған.

Осылайша бағдарламаның деректер құрылымы химиялық кинетика теңдеулерін тиімді есептеу үшін ұйымдастырылған. Барлық негізгі параметрлер массивтерде сақталады және GPU kernel функциялары арқылы параллель түрде өңделеді. Мұндай тәсіл деректерге жылдам қол жеткізуге және есептеу операцияларын тиімді орындауға мүмкіндік береді. Кинетикалық механизмдерді GPU-да тиімді есептеу үшін деректер құрылымын массивтер түрінде ұйымдастыру және жадыға қол жеткізуді оңтайландыру маңызды [61, 62].

3.2.4 GPU жадысын ұйымдастыру

CUDA негізіндегі есептеу жүйелерінде GPU жадысын тиімді ұйымдастыру жоғары өнімді параллель есептеулерді жүзеге асырудың негізгі шарттарының бірі болып табылады. Графикалық процессор архитектурасында жады бірнеше деңгейден тұратын иерархиялық құрылым түрінде ұйымдастырылған. Әрбір жады түрі өзінің көлемімен, қол жеткізу жылдамдығымен және қолдану аймағымен ерекшеленеді. Сондықтан CUDA бағдарламасын құру кезінде деректерді дұрыс жады деңгейіне орналастыру есептеу өнімділігін айтарлықтай арттыруға мүмкіндік береді.

GPU архитектурасында жадының негізгі түрлері келесілер болып табылады:

- Global memory (ғаламдық жады)
- Shared memory (ортақ жады)
- Register memory (регистр жадысы)
- Constant memory (тұрақты жады)
- Local memory (локальды жады)

Бұл жады түрлері қол жеткізу жылдамдығы мен көлемі бойынша бір-бірінен айтарлықтай ерекшеленеді. CUDA бағдарламасында есептеу тиімділігі көбінесе осы жады деңгейлерін дұрыс пайдалануына байланысты болады. CUDA архитектурасында жады иерархиясын дұрыс пайдалану kernel өнімділігіне тікелей әсер етеді, себебі global memory, shared memory және registers әртүрлі кідіріс пен өткізу қабілетіне ие [63, 64].

Ғаламдық жады GPU құрылғысының негізгі жады кеңістігі болып табылады. Бұл жады барлық CUDA ағындарына қолжетімді. Сонымен қатар CPU (host) жүйесі де осы жадыға деректерді көшіру арқылы қол жеткізе алады. Ғаламдық жадының көлемі үлкен болғанымен, оған қол жеткізу уақыты басқа жады түрлерімен салыстырғанда баяу болады.

Ұсынылған кинетикалық солвер бағдарламасында негізгі деректер массивтері ғаламдық жадыда сақталады. Оларға келесі массивтер жатады:

- химиялық компоненттердің концентрациялары

- реакция константалары
- стехиометриялық коэффициенттер
- Runge–Kutta аралық коэффициенттері

Ғаламдық жадыдағы деректерге қол жеткізу кезінде жады транзакциялары орындалады. Бұл транзакциялар белгілі бір жады блоктарымен жұмыс істейді. Егер CUDA ағындары бір-біріне жақын орналасқан деректерге қол жеткізсе, онда жады транзакциялары біріктіріледі. Мұндай жағдай **memory coalescing** деп аталады.

Жадыға тиімді қол жеткізу үшін деректердің орналасуы келесі шартты қанағаттандыруы қажет:

$$Address_i = Base + i \cdot sizeof(T) \quad (35)$$

мұндағы T — деректер типі. Мұндай орналасу кезінде GPU ағындары жадыға бір уақытта тиімді түрде қол жеткізе алады.

Ортақ жады (Shared Memory)

Ортақ жады GPU мультипроцессорының ішінде орналасқан жоғары жылдамдықты жады болып табылады. Бұл жады тек бір блокқа кіретін CUDA ағындары арасында ортақ пайдаланылады. Ортақ жадының негізгі артықшылығы — оған қол жеткізу жылдамдығы өте жоғары.

Ұсынылған кинетикалық солвер алгоритмінде ортақ жады бірнеше жағдайда қолданылады. Мысалы, бір блоктағы ағындар бірдей концентрация мәндерін қолданған кезде оларды ортақ жадыға көшіру есептеу тиімділігін арттырады. Бұл тәсіл ғаламдық жадыға қол жеткізу санын азайтады.

Ортақ жадыда деректер келесі түрде анықталуы мүмкін:

```
__shared__ double sharedData[M];
```

Бұл жағдайда блоктағы барлық ағындар sharedData массивіне қол жеткізе алады. Мұндай тәсіл ағындар арасындағы деректер алмасуын жылдам орындауға мүмкіндік береді. Shared memory блок ішіндегі ағындар арасында жылдам деректер алмасуға мүмкіндік береді және global memory-ге қатынау санын азайту арқылы есептеу өнімділігін арттырады [65, 66].

Регистр жадысы (Register Memory)

Регистр жадысы GPU-дағы ең жылдам жады түрі болып табылады. Әрбір CUDA ағыны өзінің жеке регистрлер жиынтығына ие болады. Регистрлер уақытша айнымалыларды сақтау үшін қолданылады.

Ұсынылған алгоритмде келесі айнымалылар регистрлерде сақталады:

- аралық есептеу нәтижелері
- реакция жылдамдығы
- уақытша көбейтінділер (mult1, mult2)

Мысалы, реакция жылдамдығын есептеу кезінде келесі өрнек қолданылады.

$$mult1 = \prod_{j=1}^M c_j^{left_{i,j}} \quad (36)$$

$$mult2 = \prod_{j=1}^M c_j^{right_{i,j}} \quad (37)$$

Бұл көбейтінділер регистрлерде сақталған кезде есептеу жылдамдығы айтарлықтай артады.

Тұрақты жады (Constant Memory)

Тұрақты жады тек оқуға арналған жады болып табылады. Бұл жады барлық CUDA ағындарына қолжетімді және арнайы кэш арқылы жұмыс істейді. Егер барлық ағындар бірдей деректерге жүгінсе, тұрақты жады өте тиімді жұмыс істейді.

Кинетикалық есептерде тұрақты жады келесі параметрлерді сақтау үшін қолданылуы мүмкін:

- реакция константалары
- физикалық тұрақтылар
- модель параметрлері

Мысалы:

```
__constant__ double kf_const[N];
```

Бұл тәсіл барлық ағындарға бірдей деректерді тиімді таратуға мүмкіндік береді.

Локальды жады (Local Memory)

Локальды жады әрбір CUDA ағынына жеке тиесілі жады болып табылады. Бұл жады негізінен регистрлер жетіспеген жағдайда қолданылады. Егер компилятор айнымалыларды регистрде сақтай алмаса, олар локальды жадыға орналастырылады.

Локальды жады физикалық тұрғыдан ғаламдық жадыда орналасады, сондықтан оған қол жеткізу салыстырмалы түрде баяу болады. Сондықтан CUDA бағдарламаларын жазу кезінде регистрлерді тиімді пайдалану өте маңызды.

GPU жадысын ұйымдастыру алгоритмі

GPU жадысын ұйымдастыру бағдарламаның бастапқы кезеңінде орындалады. Бұл процесс жадыны бөлу, деректерді көшіру және kernel функцияларын іске қосу үшін қажетті құрылымдарды дайындауды қамтиды.

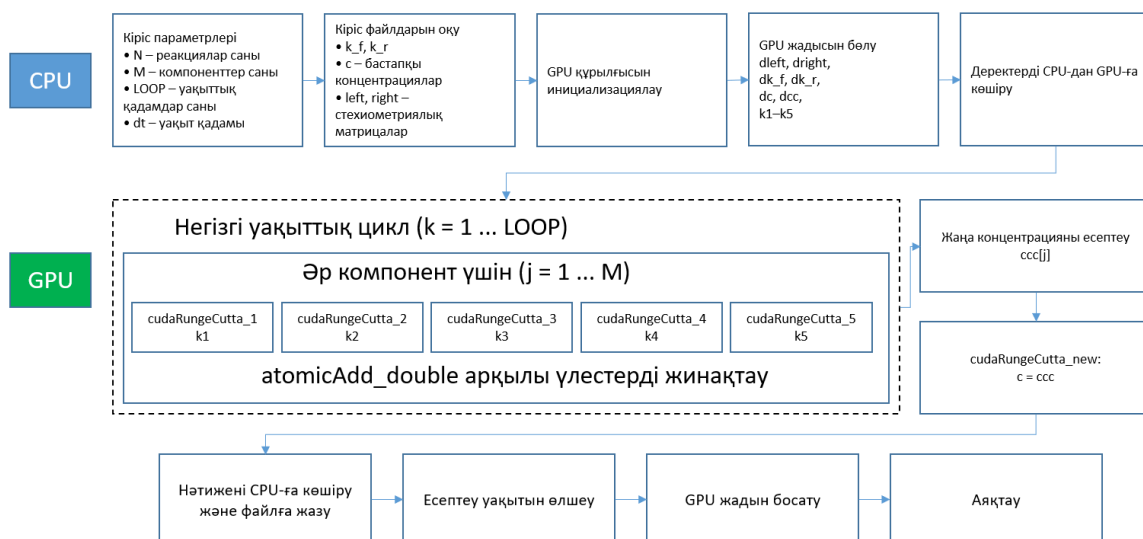
GPU жадысын ұйымдастыру бірнеше кезеңнен тұрады. Алдымен барлық қажетті массивтер үшін жады бөлінеді. Кейін бастапқы деректер GPU жадысына көшіріледі. Осыдан кейін kernel функциялары арқылы есептеу жүргізіледі. Есептеу аяқталғаннан кейін нәтижелер CPU жадысына қайта көшіріледі.

Осылайша GPU жадысын дұрыс ұйымдастыру CUDA негізіндегі кинетикалық солвердің тиімді жұмыс істеуін қамтамасыз етеді. Өртүрлі жады деңгейлерін дұрыс пайдалану есептеу жылдамдығын арттыруға және жады

ресурстарын тиімді қолдануға мүмкіндік береді. Ғаламдық жады негізгі деректерді сақтау үшін қолданылса, ортақ жады ағындар арасындағы деректер алмасуды жылдамдатуға мүмкіндік береді. Регистрлер уақытша есептеулер үшін пайдаланылады, ал тұрақты жады жиі қолданылатын параметрлерді тиімді сақтау үшін қолданылады.

3.3 Runge–Kutta әдісінің параллель іске асырылуы

Химиялық кинетика теңдеулерін шешу үшін сандық интегралдау әдістері кеңінен қолданылады. Бұл теңдеулер жүйесі әдетте қатты байланысқан және сызықтық емес дифференциалдық теңдеулерден тұрады. Мұндай жүйелерді аналитикалық түрде шешу көп жағдайда мүмкін емес, сондықтан сандық әдістерді қолдану қажет. Осы жұмыста дифференциалдық теңдеулер жүйесін шешу үшін жоғары дәлдікке ие **Runge–Kutta әдісі** қолданылады (сурет 17). Runge–Kutta әдістері химиялық кинетика теңдеулерін уақыт бойынша интегралдау кезінде кеңінен қолданылады, ал GPU архитектурасында олардың кезеңдерін параллель орындау есептеу уақытын қысқартады [67, 68].



Сурет 17 – CPU және GPU өзара әрекеттесуі негізінде CUDA кинетикалық солверінің орындалу алгоритмі

Runge–Kutta әдісі — бастапқы шарттары берілген қарапайым дифференциалдық теңдеулерді сандық интегралдау үшін қолданылатын көпсатылы әдістердің бірі. Бұл әдіс уақыт бойынша интегралдау кезінде бірнеше аралық есептеулерді орындау арқылы шешімнің дәлдігін арттыруға мүмкіндік береді. Химиялық кинетика есептерінде бұл әдіс реакция жылдамдықтарының өзгерісін уақыт бойынша дәл анықтауға мүмкіндік береді.

Жалпы түрде химиялық кинетика теңдеулер жүйесі келесі түрде жазылады.

$$\frac{dC_k}{dt} = F_k(C_1, C_2, \dots, C_M) \quad (38)$$

мұндағы C_k — k -ші химиялық компоненттің концентрациясы, ал F_k — химиялық реакциялар әсерінен концентрацияның өзгеру функциясы.

Химиялық кинетика теңдеулерінде концентрациялардың өзгерісі барлық реакциялардың қосындысымен анықталады. Бұл келесі түрде өрнектеледі.

$$\frac{dC_k}{dt} = \sum_{i=1}^N (v_{ik}^L - v_{ik}^R) r_i \quad (39)$$

мұндағы N — реакциялар саны,

v_{ik}^L — реагенттердің стехиометриялық коэффициенттері,

v_{ik}^R — өнімдердің стехиометриялық коэффициенттері,

r_i — i -ші реакцияның жылдамдығы.

Runge–Kutta әдісінің негізгі идеясы — уақыт қадамы ішінде бірнеше аралық мәндерді есептеу арқылы шешімді жақсарту. Бұл әдісте бірнеше аралық коэффициенттер анықталады. Осы жұмыста қолданылатын схема бес аралық коэффициенттен тұрады.

Runge–Kutta әдісінің жалпы формуласы келесі түрде жазылады.

$$C_{n+1} = C_n + dt \cdot \Phi(k_1, k_2, k_3, k_4, k_5) \quad (40)$$

мұндағы dt — уақыт қадамы, ал Φ — аралық коэффициенттердің комбинациясы.

Runge–Kutta әдісінде бірінші коэффициент келесі түрде есептеледі.

$$k_1 = F(C_n) \quad (41)$$

Екінші коэффициент концентрацияның аралық мәнін пайдаланып есептеледі.

$$k_2 = F\left(C_n + \frac{dt}{2} k_1\right) \quad (42)$$

Үшінші коэффициент келесі түрде анықталады.

$$k_3 = F\left(C_n + \frac{dt}{2} k_2\right) \quad (43)$$

Төртінші коэффициент:

$$k_4 = F(C_n + dt \cdot k_3) \quad (44)$$

Ал бесінші коэффициент уақыт қадамының соңындағы функция мәнін сипаттайды.

$$k_5 = F(C_n + dt \cdot k_4) \quad (45)$$

Барлық коэффициенттер есептелгеннен кейін концентрацияның жаңа мәні анықталады.

CUDA технологиясын қолданған кезде Runge–Kutta әдісінің барлық аралық есептеулері GPU архитектурасында параллель түрде орындалады. Бұл жағдайда әрбір CUDA ағыны белгілі бір реакция немесе компонент үшін есептеу жүргізеді.

Параллель есептеу кезінде келесі принцип қолданылады:

- әрбір **thread** жеке реакцияны өңдейді
- бір **block** бірнеше реакцияны есептейді
- барлық **blocks** реакциялар жүйесін толық өңдейді

Бұл тәсіл үлкен реакциялар жүйесін өте жылдам есептеуге мүмкіндік береді.

Runge–Kutta әдісінің әрбір кезеңі жеке CUDA kernel функциясы ретінде іске асырылады. Kernel іске қосылған кезде GPU көптеген ағындарды бір уақытта орындайды. Әрбір ағын реакция жылдамдығын есептейді және концентрация өзгерісіне өз үлесін қосады.

3.4 Параллель есептеудің артықшылықтары

Runge–Kutta әдісін GPU архитектурасында параллель орындаудың бірнеше маңызды артықшылықтары бар:

1. **Есептеу уақытын қысқарту** Бір уақытта мыңдаған CUDA ағындары жұмыс істей алады, бұл үлкен реакциялар жүйесін жылдам есептеуге мүмкіндік береді.

2. **Жоғары масштабталу мүмкіндігі** Реакциялар саны артқан сайын GPU ағындарының саны да көбейеді.

3. **Жадыны тиімді пайдалану** Параллель есептеу кезінде әрбір ағын өз деректерін жеке өңдейді.

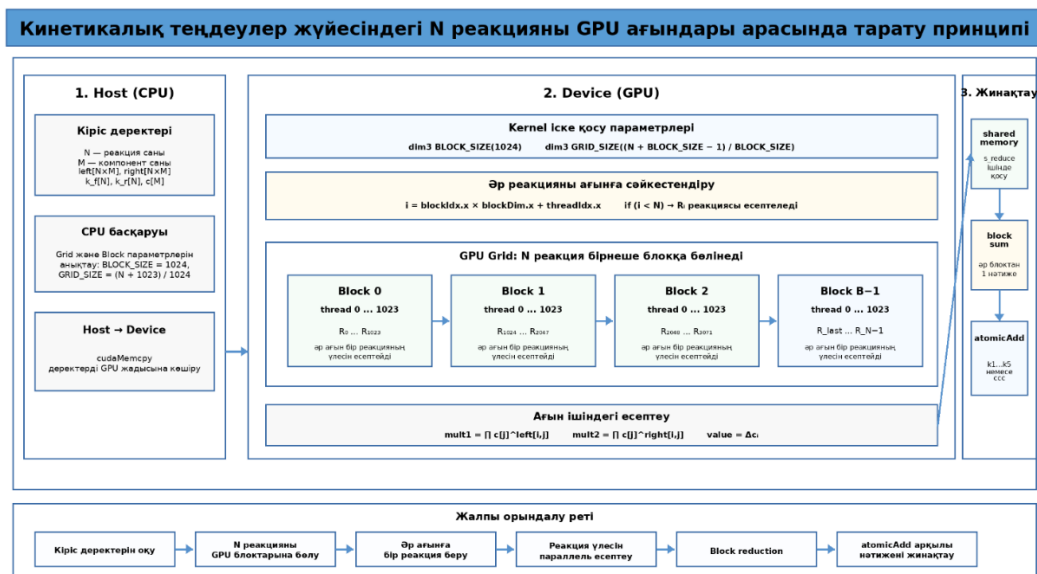
4. **Үлкен модельдерді есептеу мүмкіндігі** Жүздеген немесе мыңдаған реакциялары бар химиялық жүйелерді модельдеуге мүмкіндік береді.

Осылайша Runge–Kutta әдісін CUDA архитектурасында параллель іске асыру химиялық кинетика теңдеулерін есептеуді айтарлықтай жылдамдатады. GPU-дың көпағынды құрылымы әрбір реакцияны тәуелсіз түрде есептеуге мүмкіндік береді. Бұл тәсіл үлкен реакциялар жүйесін модельдеу кезінде есептеу тиімділігін бірнеше есе арттыруға мүмкіндік береді. Параллель есептеудің тиімділігі реакциялар саны артқан сайын айқын байқалады, себебі тәуелсіз есептеу операциялары GPU ағындары арасында үлестіріледі [69].

3.4.1 Атомарлық операцияларды қолдану

CUDA архитектурасында параллель есептеулер көптеген есептеу ағындарының (threads) бір уақытта орындалуына негізделеді. Бұл тәсіл есептеу өнімділігін айтарлықтай арттыруға мүмкіндік береді. Дегенмен, параллель есептеулер кезінде бірнеше ағын бір уақытта бір деректер аймағына қол жеткізген жағдайда синхронизация мәселелері туындауы мүмкін. Әсіресе бірнеше ағын бір уақытта бір жады ұяшығына мән жазуға тырысқан жағдайда

нәтижелердің дұрыстығы бұзылуы ықтимал. Мұндай жағдай **race condition** деп аталады (сурет 18).



Сурет 18 – GPU ағындары арасында реакцияларды тарату сұлбасы

Race condition кезінде бірнеше есептеу ағындары бір айнымалының мәнін бір уақытта өзгертуге тырысады. Егер бұл операциялар дұрыс синхрондалмаса, соңғы нәтиже дұрыс болмайды, себебі кейбір жазу операциялары бір-бірін басып кетуі мүмкін. Химиялық кинетика есептерінде бұл мәселе жиі кездеседі, себебі бір химиялық компоненттің концентрациясына бірнеше реакция бір уақытта әсер етеді.

Химиялық кинетика теңдеулерінде компонент концентрациясының өзгерісі барлық реакциялар әсерінің қосындысы арқылы анықталады. Әрбір реакция белгілі бір компоненттің концентрациясына өз үлесін қосады. Егер CUDA бағдарламасында әрбір реакцияны жеке есептеу ағыны өңдейтін болса, онда бірнеше ағын бір уақытта бір компоненттің концентрациясының өзгерісін есептеуге қатысады. Осындай жағдайда барлық ағындардың нәтижелерін дұрыс жинақтау үшін арнайы синхронизация механизмін қолдану қажет.

CUDA платформасында мұндай синхронизацияны қамтамасыз ету үшін **атомарлық операциялар (atomic operations)** қолданылады. Атомарлық операция — бұл бірнеше ағын бір уақытта бір жады ұяшығына қол жеткізген жағдайда операцияның толық орындалуын қамтамасыз ететін арнайы механизм. Мұндай операция орындалған кезде басқа ағындар осы жады ұяшығына қол жеткізуді операция аяқталғанша күтіп тұрады. Осылайша деректердің бұзылуының алдын алуға болады.

CUDA кітапханасында бірнеше атомарлық операциялар бар. Олардың ішінде кең таралғандары:

- atomicAdd()
- atomicSub()
- atomicExch()
- atomicMin()

- `atomicMax()`

Кинетикалық солвер алгоритмінде негізінен **`atomicAdd()`** операциясы қолданылады. Бұл операция белгілі бір жады ұяшығына жаңа мәнді атомарлық түрде қосуға мүмкіндік береді.

Мысалы, реакциялар жүйесінде әрбір CUDA ағыны белгілі бір реакцияның үлесін есептегеннен кейін оны концентрация өзгерісінің жалпы мәніне қосуы қажет. Бұл операция келесі түрде жүзеге асырылады.

`atomicAdd(&dc[k], value);`

Мұнда $dc[k]$ — k -ші компонент концентрациясының өзгерісін сақтайтын массив элементі, ал `value` — белгілі бір реакцияның үлесі.

Атомарлық операцияны қолдану кезінде барлық ағындар бір айнымалыға қауіпсіз түрде мән қоса алады. Бұл жағдайда операциялардың орындалу реті маңызды болмайды, себебі әрбір операция толық орындалады [70].

CUDA архитектурасында атомарлық операциялар аппараттық деңгейде жүзеге асырылады. Бұл операциялар арнайы синхронизация механизмдері арқылы орындалады. Егер бірнеше ағын бір уақытта бір жады ұяшығына қол жеткізгісі келсе, GPU аппараттық жоспарлаушысы оларды кезекпен орындайды.

Атомарлық операцияларды қолдану параллель есептеулердің дұрыстығын қамтамасыз етеді. Алайда бұл операциялар белгілі бір есептеу шығындарын талап етеді. Себебі бірнеше ағын бір уақытта бір жады ұяшығына қол жеткізген жағдайда олардың кейбіреуі операцияның аяқталуын күтуге мәжбүр болады. Сондықтан CUDA бағдарламаларын жазу кезінде атомарлық операцияларды мүмкіндігінше аз қолдану ұсынылады.

Ұсынылған кинетикалық солвер алгоритмінде атомарлық операциялар реакция үлестерін жинақтау кезінде қолданылады. Әрбір CUDA ағыны реакция жылдамдығын есептегеннен кейін концентрация өзгерісінің жалпы мәніне өз үлесін қосады.

Әрбір CUDA ағыны осы реакция жылдамдығын есептеп болғаннан кейін концентрация өзгерісін анықтайды.

Алынған мән жалпы концентрация өзгерісіне қосылады. Бұл операция атомарлық түрде орындалады.

Әрбір CUDA ағыны реакция жылдамдығын есептеп болғаннан кейін концентрация өзгерісінің үлесін атомарлық операция арқылы жалпы массивке қосады. Осылайша барлық реакциялар әсері дұрыс жинақталады.

CUDA архитектурасында тағы бір маңызды мәселе — **double типіндегі айнымалылармен жұмыс істеу**. Кейбір GPU архитектураларында стандартты `atomicAdd()` функциясы тек `float` типі үшін ғана қолжетімді болуы мүмкін. Сондықтан `double` типіндегі айнымалылар үшін арнайы атомарлық операцияны іске асыру қажет.

Бұл жағдайда **compare-and-swap (CAS)** механизмі қолданылады. CAS операциясы жады ұяшығының ағымдағы мәнін тексеріп, егер ол күтілетін мәнге тең болса ғана жаңа мәнді жазады. Егер мән өзгеріп кеткен болса, операция қайта орындалады. Бұл тәсіл `double` типіндегі айнымалылармен қауіпсіз түрде жұмыс істеуге мүмкіндік береді.

Осылайша атомарлық операцияларды қолдану CUDA негізіндегі кинетикалық солвердің маңызды элементтерінің бірі болып табылады. Бұл операциялар параллель есептеулер кезінде деректердің тұтастығын сақтауға мүмкіндік береді. Әсіресе көптеген реакциялар бір химиялық компонентке әсер ететін жүйелерде атомарлық операциялар нәтижелердің дұрыстығын қамтамасыз етеді.

Атомарлық операцияларды дұрыс қолдану нәтижесінде барлық CUDA ағындарының есептеу нәтижелері дұрыс жинақталады және химиялық компоненттердің концентрацияларының нақты мәндері анықталады. Сонымен қатар бұл тәсіл GPU архитектурасының параллель мүмкіндіктерін толық пайдалануға мүмкіндік береді.

3.5 Нәтижелерді сақтау

CUDA негізіндегі кинетикалық солверде есептеу нәтижелерін сақтау ғылыми есептеулердің маңызды кезеңдерінің бірі болып табылады. Химиялық кинетика теңдеулерін уақыт бойынша интегралдау барысында алынған нәтижелер кейінгі талдау, визуализация және модельдің дұрыстығын тексеру үшін қолданылады. Сондықтан есептеу нәтижелерін дұрыс ұйымдастыру, оларды жадыда сақтау және сыртқы файлдарға жазу алгоритмінің толық жұмыс істеуінің маңызды бөлігі болып табылады. GPU-да есептелген нәтижелерді CPU жадысына көшіру және оларды құрылымдалған файлдық форматта сақтау кейінгі талдау мен визуализация үшін қажет [71].

Ұсынылған бағдарламалық жүйеде нәтижелерді сақтау бірнеше кезеңнен тұрады. Бұл кезеңдерге GPU құрылғысында есептелген деректерді CPU жадысына көшіру, нәтижелерді өңдеу және файлдарға жазу жатады. Әрбір уақыт қадамынан кейін алынған концентрация мәндері сақталып отырады. Бұл тәсіл химиялық жүйенің уақыт бойынша эволюциясын бақылауға мүмкіндік береді.

Химиялық кинетика теңдеулерін шешу нәтижесінде негізгі есептелетін параметр — химиялық компоненттердің концентрациялары болып табылады. Егер жүйеде M химиялық компонент болса, онда әрбір уақыт қадамында M концентрация мәні анықталады.

CUDA бағдарламасында есептеулер GPU құрылғысында орындалғанымен, нәтижелерді сақтау және талдау әдетте CPU жағында жүзеге асырылады. Сондықтан есептеу аяқталғаннан кейін немесе әрбір уақыт қадамынан кейін нәтижелер GPU жадысынан CPU жадысына көшіріледі.

Бұл операция CUDA кітапханасының **`cudaMemcpy()`** функциясы арқылы жүзеге асырылады. Бұл функция GPU және CPU арасындағы деректер алмасуды қамтамасыз етеді.

Нәтижелерді көшірудің жалпы форматы келесі түрде жазылады:

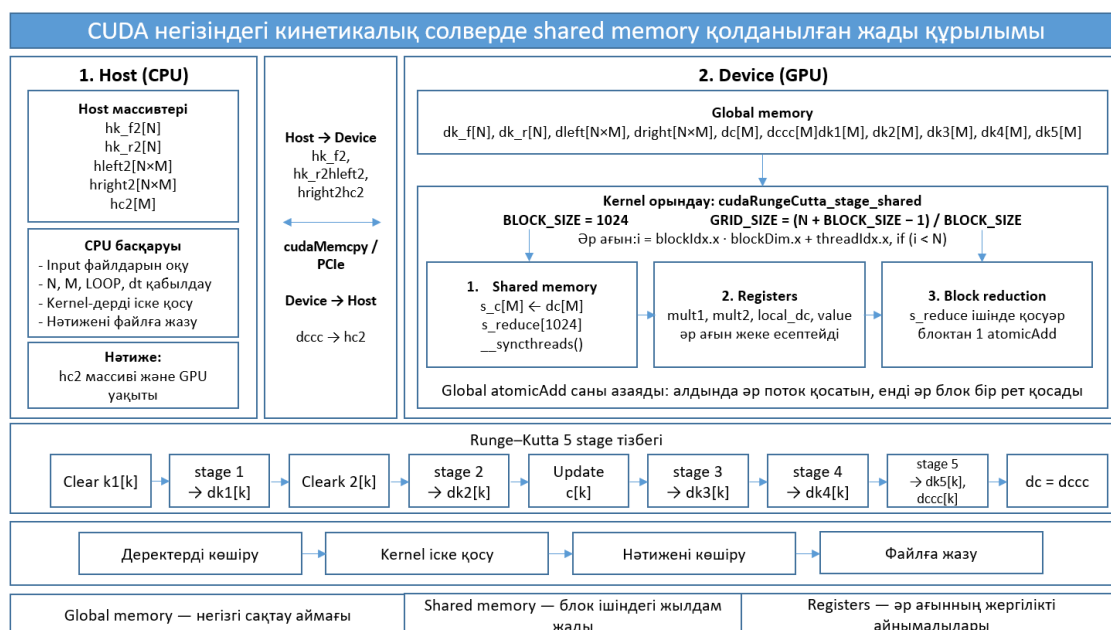
`cudaMemcpy(c_host, c_device, size, cudaMemcpyDeviceToHost)`

Мұнда

`c_device` — GPU жадысында орналасқан концентрациялар массиві,

`c_host` — CPU жадысында орналасқан қабылдаушы массив.

Бұл операция орындалғаннан кейін барлық есептеу нәтижелері CPU жадысында қолжетімді болады.



Сурет 19 – Shared memory қолданылған CUDA жады құрылымы

Сурет 19-да CUDA негізіндегі кинетикалық солверде shared memory қолдану арқылы GPU жадысын ұйымдастыру сұлбасы көрсетілген. Бұл сұлба CPU жағындағы бастапқы деректерді дайындаудан бастап, оларды GPU жадысына көшіру, CUDA ядросын іске қосу, Runge–Kutta әдісінің кезеңдерін орындау, нәтижені қайта көшіру және файлға жазу кезеңдерін қамтиды.

Есептеу процесі алдымен Host (CPU) жағында орындалатын дайындық кезеңінен басталады. Бұл кезеңде реакция жылдамдығының тұрақтылары, стехиометриялық матрицалар және бастапқы концентрациялар сияқты негізгі массивтер қалыптастырылады. Атап айтқанда, $hk_f2[N]$, $hk_r2[N]$, $hleft2[N \times M]$, $hright2[N \times M]$ және $hc2[M]$ массивтері CPU жадында сақталады. Сонымен қатар CPU басқарушы бөлігі кіріс файлдарын оқып, реакциялар саны N , компоненттер саны M , уақыт қадамы және басқа да есептеу параметрлерін анықтайды.

Одан кейін дайындалған деректер Host → Device бағыты бойынша GPU жадысына көшіріледі. Бұл операция `cudaMemcpy` және PCIe шинасы арқылы жүзеге асырылады. GPU жағында деректер алдымен global memory аймағында сақталады. Global memory негізгі сақтау аймағы ретінде қолданылады және онда $dk_f[N]$, $dk_r[N]$, $dleft[N \times M]$, $dright[N \times M]$, $dc[M]$, сондай-ақ Runge–Kutta әдісінің аралық массивтері $dk1[M]$, $dk2[M]$, $dk3[M]$, $dk4[M]$, $dk5[M]$ орналасады.

Негізгі есептеу бөлігі Device (GPU) деңгейінде орындалады. CUDA ядросы `cudaRungeKutta_stage_shared` түрінде іске қосылады. Мұнда $BLOCK_SIZE = 1024$, ал тор өлшемі $GRID_SIZE = (N + BLOCK_SIZE - 1) / BLOCK_SIZE$ формуласы арқылы анықталады. Әрбір GPU ағыны бір реакцияны өңдеуге жауап береді, яғни ағын индексі $i = blockIdx.x \cdot blockDim.x + threadIdx.x$ түрінде есептеледі. Егер $i < N$ шарты орындалса, онда осы ағын сәйкес реакция бойынша есептеу жүргізеді.

Бұл сұлбаның негізгі ерекшелігі — shared memory қолданылуы. Әр блок ішінде концентрациялар массивінің көшірмесі $s_c[M]$ түрінде shared memory-ге жүктеледі. Shared memory global memory-ге қарағанда жылдамырақ болғандықтан, әр реакцияны есептеу кезінде концентрация мәндеріне қатынау уақыты азаяды. Бұл әсіресе реакциялар саны көп және бірдей концентрация мәндері бірнеше рет қолданылатын жағдайда есептеу өнімділігін арттырады.

Есептеу барысында әрбір ағын реакцияға қатысты локальды шамаларды registers ішінде сақтайды. Registers әр ағынға тиесілі ең жылдам локальды жады болып табылады. Онда mult1, mult2, local_dc және value сияқты аралық айнымалылар орналасады. Бұл тәсіл global memory-ге артық қатынауды азайтып, CUDA ядросының орындалуын жеделдетеді.

Әр реакция бойынша алынған үлестер кейін блок ішінде жинақталады. Бұл кезеңде block reduction әдісі қолданылады. Алдымен әр ағын есептеген мәндер блок ішінде біріктіріледі, содан кейін ғана бір рет atomicAdd операциясы орындалады. Мұндай тәсіл global atomicAdd операцияларының санын азайтады. Яғни бастапқы нұсқада әр ағын global memory-ге жеке-жеке қосу операциясын орындаса, shared memory қолданылған нұсқада алдымен блок ішінде жинақтау жүргізіліп, global memory-ге тек қорытынды мән қосылады. Бұл GPU есептеу уақытын едәуір қысқартуға мүмкіндік береді.

Runge–Kutta әдісінің бес кезеңі сұлбада жеке көрсетілген. Алдымен k1 массиві тазартылып, бірінші кезеңде $dk1[k]$ есептеледі. Кейін екінші кезеңде k2, үшінші кезеңде k3, төртінші кезеңде k4, бесінші кезеңде k5 есептеледі. Соңында жаңа концентрация мәні $dc = d_{cc}$ түрінде жаңартылады. Осылайша әр уақыт қадамында концентрациялар Runge–Kutta әдісінің бес сатылы схемасы бойынша есептеледі.

Есептеу аяқталғаннан кейін нәтиже Device → Host бағыты бойынша CPU жадысына қайта көшіріледі. Нәтижелік массив ретінде hc2 және GPU есептеу уақыты алынады. Соңғы кезеңде нәтижелер файлға жазылып, GPU жадысы босатылады.

Осылайша, сурет 19-да көрсетілген жадыны ұйымдастыру сұлбасы CUDA негізіндегі кинетикалық солвердің өнімділігін арттыруға бағытталған. Global memory негізгі деректерді сақтау үшін, shared memory блок ішіндегі жылдам қатынау үшін, ал registers әр ағынның локальды есептеулері үшін қолданылады. Мұндай иерархиялық жады құрылымы жадыға қатынау кідірістерін азайтып, реакциялар саны көп кинетикалық теңдеулер жүйесін GPU-да тиімді есептеуге мүмкіндік береді.

Химиялық жүйенің уақыт бойынша эволюциясын талдау үшін әрбір уақыт қадамында алынған концентрация мәндерін сақтау қажет. Бұл үшін арнайы нәтижелер массиві қолданылады.

$$Result[t][k] = C_k(t) \quad 46$$

мұнда, t — уақыт қадамы, k — химиялық компонент индексі.

Бұл массивте барлық уақыт қадамдары үшін концентрациялар сақталады. Мұндай тәсіл химиялық реакциялардың динамикасын толық сипаттауға мүмкіндік береді.

Есептеу нәтижелерін кейінгі талдау үшін файлдарға сақтау қажет. Бағдарламада нәтижелер мәтіндік немесе кестелік форматта сақталуы мүмкін. Әдетте ғылыми есептеулерде нәтижелер **CSV**, **TXT** немесе **DAT** форматтарында жазылады.

Бірінші бағанда уақыт мәндері, ал келесі бағандарда химиялық компоненттердің концентрациялары көрсетіледі.

Нәтижелерді файлға жазу бағдарламаның соңғы кезеңдерінің бірі болып табылады. Бұл процесс CPU жағында орындалады.

Нәтижелерді сақтау процесінің негізгі кезеңдерін көрсетеді. Алдымен барлық деректер GPU жадысынан CPU жадысына көшіріледі. Содан кейін уақыт бойынша алынған концентрациялар арнайы нәтижелер массивіне жазылады. Соңында барлық нәтижелер файлға сақталады.

Нәтижелерді сақтау кезеңінде есептеу уақыты да өлшенуі мүмкін. Бұл GPU алгоритмінің тиімділігін бағалау үшін маңызды. CUDA платформасында есептеу уақытын өлшеу үшін арнайы уақыт өлшеу механизмдері қолданылады.

Нәтижелерді сақтау кезінде бірнеше маңызды факторларды ескеру қажет:

1. **Жадыны тиімді пайдалану** Нәтижелер массивтері алдын ала бөлінген жадыда сақталуы керек.
2. **Деректерді көшіру санын азайту** GPU мен CPU арасында деректер алмасуды мүмкіндігінше аз орындау қажет.
3. **Файл жазу жылдамдығы** Үлкен деректер көлемін сақтау кезінде файл жазу операциялары есептеу уақытын арттыруы мүмкін.
4. **Нәтижелерді құрылымдау** Нәтижелер кейінгі талдау үшін ыңғайлы форматта сақталуы тиіс.

Осылайша нәтижелерді сақтау CUDA негізіндегі кинетикалық солвердің маңызды кезеңдерінің бірі болып табылады. Бұл кезеңде GPU құрылғысында есептелген концентрация мәндері CPU жадысына көшіріліп, арнайы массивтерде сақталады. Кейін бұл нәтижелер файлдарға жазылып, ғылыми талдау жүргізу үшін пайдаланылады.

Нәтижелерді дұрыс ұйымдастыру химиялық жүйенің уақыт бойынша өзгерісін толық зерттеуге мүмкіндік береді. Сонымен қатар нәтижелерді сақтау есептеу алгоритмінің өнімділігін бағалауға және модельдің дұрыстығын тексеруге мүмкіндік береді.

Кесте 2-де Aramco_Mech механизмі негізінде реакциялар саны артқан сайын CPU және GPU есептеу уақыттарының өзгерісі көрсетілген. Есептеу тәжірибесінде уақыт қадамы $dt = 1,0 \cdot 10^{-19}$ с, ал жалпы есептеу уақыты $1000 \cdot dt = 1,0 \cdot 10^{-16}$ с болып алынды. Сандық интегралдау үшін бесінші дәрежелі Рунге–Кутта әдісі қолданылды. Есептеу тәжірибесі Intel Core i7-7700K процессоры, 32 ГБ RAM және NVIDIA GeForce Titan X графикалық адаптері бар есептеу жүйесінде жүргізілді.

Алынған нәтижелер реакциялар саны артқан сайын CPU арқылы есептеу

уақытының едәуір өсетінін көрсетеді. Мысалы, реакциялар саны 118 болған жағдайда CPU уақыты 39 с 81 мс құраса, реакциялар саны 1006-ға жеткенде бұл көрсеткіш 1 сағ 53 мин 52 с 52 мс-қа дейін артты. Бұл кинетикалық механизм күрделенген сайын тізбекті есептеулерге түсетін жүктеменің айтарлықтай көбейетінін көрсетеді.

GPU негізіндегі есептеулерде де есептеу уақыты өседі, алайда бұл өсім CPU-мен салыстырғанда әлдеқайда баяу. Реакциялар саны 118 болған кезде GPU уақыты 20 с 204 мс болса, 1006 реакция үшін GPU уақыты 6 мин 30 с 403 мс болды. Бұл графикалық процессор архитектурасының параллель есептеулерді тиімді орындауға мүмкіндік беретінін көрсетеді.

Жылдамдату коэффициенті CPU уақытының GPU уақытына қатынасы арқылы анықталды:

$$S = \frac{T_{CPU}}{T_{GPU}} \quad 47$$

мұндағы S — жылдамдату коэффициенті, T_{CPU} — орталық процессордағы есептеу уақыты, ал T_{GPU} — графикалық процессордағы есептеу уақыты.

Кестедегі нәтижелер бойынша реакциялар саны 118 болған жағдайда жылдамдату коэффициенті 1,93-ке тең болды. Ал реакциялар саны 1006-ға дейін артқанда жылдамдату коэффициенті 17,5-ке жетті. Бұл GPU-да параллель есептеуді қолдану әсіресе үлкен кинетикалық механизмдер үшін тиімді екенін дәлелдейді.

Осылайша, жүргізілген есептеу тәжірибесі химиялық реакциялар саны көбейген сайын GPU негізіндегі кинетикалық солвердің артықшылығы айқындала түсетінін көрсетті. Реакциялар саны артқан кезде есептеулердің көп бөлігі тәуелсіз операциялар түрінде орындалатындықтан, оларды CUDA ядролары арқылы бір мезгілде өңдеу жалпы есептеу уақытын айтарлықтай қысқартуға мүмкіндік береді.

Дайындалған бағдарламалық жасақтаманы пайдалана отырып, орталық және графикалық процессордағы есептеуге кеткен уақытқа салыстырмалы түрде болжам жасалынды. Нәтижесінде, графикалық карталарды пайдалана отырып есептеулер жүргізілген процессор арқылы уақыттан ұтыс алынды [72, 73].

Кесте 2. Механизмнің өлшеміне байланысты есептеу уақытының өзгерісі

Элементтер саны	Реакция саны	CPU уақыты	GPU уақыты	Жылдамдату
33	118	39 с 81 мс	20 с 204 мс	1,93
39	191	1 мин 27 с 735 мс	29 с 619 мс	2,96
52	245	3 мин 19 с 169 мс	49 с 747 мс	4,00
61	301	5 мин 36 с 749 мс	1 мин 7 с 557 мс	4,98
66	376	8 мин 11 с 811 мс	1 мин 23 с 188 мс	5,91
84	448	15 мин 46 с 439 мс	2 мин 5 с 339 мс	7,55
103	565	29 мин 55 с 548 мс	3 мин 6 с 175 мс	9,64

162	1006	1 сағ 53 мин 52 с 52 мс	6 мин 30 с 403 мс	17,5
-----	------	----------------------------	-------------------	------

Есептеу тәжірибелері келесі техникалық сипаттамалары бар есептеу жүйесінде жүргізілді: AMD Ryzen 9 5950X 16-Core Processor орталық процессоры, 128 ГБ RAM жедел жады және NVIDIA GeForce RTX 4080 Ti графикалық адаптері қолданылды. Бұл техникалық конфигурация CPU және GPU негізіндегі есептеу уақыттарын салыстыруға, сондай-ақ GPU жадысының әртүрлі түрлерін — global memory және shared memory — қолданудың өнімділікке әсерін бағалауға мүмкіндік берді.

Осы техникалық сипаттамалар негізінде алынған нәтижелер Кесте 2-де келтірілген. Кестедегі мәліметтерді талдау GPU жадысын тиімді ұйымдастырудың есептеу өнімділігіне тікелей әсер ететінін көрсетеді.

Кесте 2-де келтірілген нәтижелерді талдау GPU жадысын тиімді ұйымдастырудың есептеу өнімділігіне тікелей әсер ететінін көрсетеді. Global memory қолданылған кезде барлық деректер графикалық процессордың жаһандық жадысынан оқылады. Бұл жад көлемі үлкен болғанымен, оған қатынау жылдамдығы shared memory-мен салыстырғанда төмен. Сондықтан реакциялар саны артқан сайын global memory арқылы орындалатын есептеулерде жадыға қатынау кідірістері жалпы есептеу уақытына көбірек әсер етеді.

Кесте мәліметтері бойынша shared memory қолданылған жағдайда есептеу уақыты барлық қарастырылған механизмдер үшін global memory-мен салыстырғанда азаяды. Бұл әсіресе реакциялар саны көп болған жағдайда айқын байқалады. Мысалы, 2716 реакциядан тұратын ең күрделі жағдайда global memory қолданылған GPU есептеу уақыты 21 мин 10 с 591 мс болса, shared memory қолданылған кезде бұл көрсеткіш 10 мин 51 с 585 мс-қа дейін қысқарған.

Осы нәтижелер Кесте 2-де берілген жылдамдату коэффициенттерімен де расталады. Реакциялар саны 118 болған кезде shared memory негізіндегі жылдамдату 3,48 болса, 2716 реакция үшін бұл көрсеткіш 154,89-ға дейін артқан. Бұл күрделі химиялық кинетикалық механизмдерде shared memory қолдану CUDA ядроларының есептеу ресурстарын тиімді пайдалануға мүмкіндік беретінін дәлелдейді.

Кесте 3 – Global memory және shared memory қолданылған кездегі CPU және GPU есептеу уақыттарының салыстырмалы нәтижелері

Элемент саны	Реакция саны	CPU уақыты	GPU уақыты, global memory	GPU уақыты, shared memory	Global жылдамдату	Shared жылдамдату
33	118	24 с 426 мс	8 с 784 мс	7 с 027 мс	2,78	3,48
39	191	54 с 834 мс	12 с 617 мс	9 с 705 мс	4,35	5,65
52	245	2 мин 4 с 418 мс	21 с 629 мс	16 с 021 мс	5,75	7,77

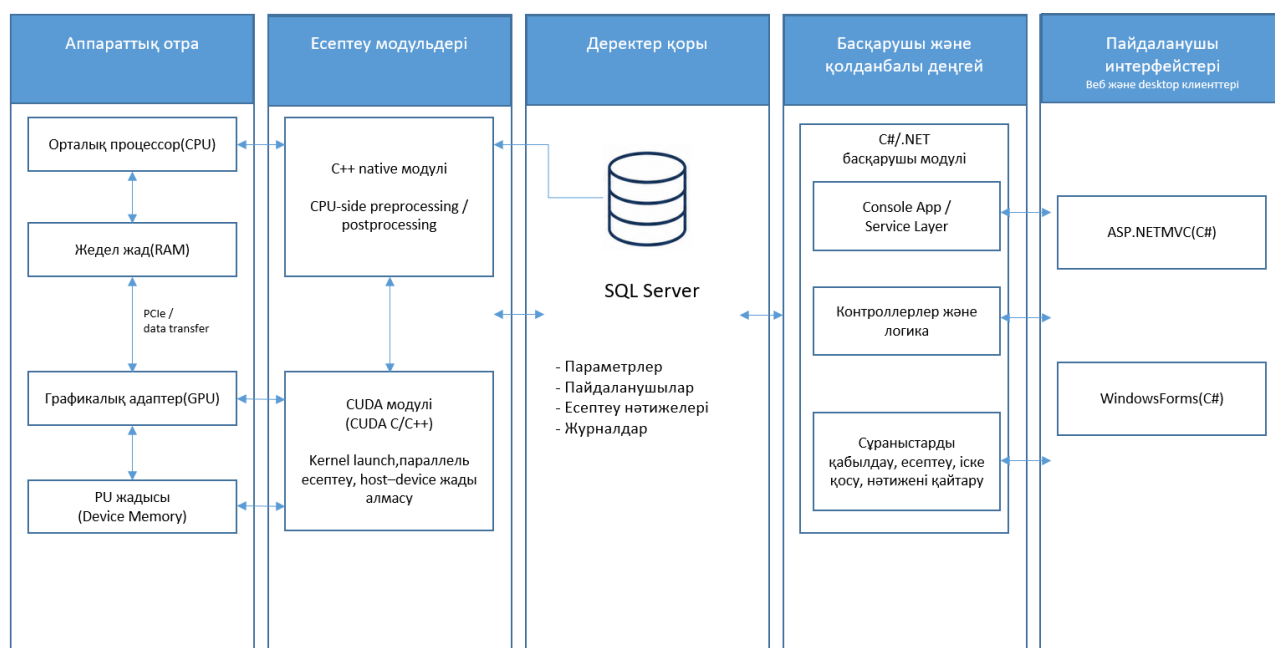
61	301	3 мин 30 с 468 мс	29 с 373 мс	20 с 981 мс	7,17	10,03
66	376	5 мин 7 с 382 мс	36 с 169 мс	24 с 944 мс	8,50	12,32
84	448	9 мин 51 с 524 мс	54 с 495 мс	36 с 330 мс	10,85	16,28
103	565	18 мин 42 с 217 мс	1 мин 20 с 946 мс	50 с 591 мс	13,86	22,18
162	1006	1 сағ 11 мин 10 с 032 мс	2 мин 49 с 740 мс	1 мин 36 с 994 мс	25,16	44,02
493	2716	1 тәул 4 сағ 2 мин 4 с 796 мс	21 мин 10 с 591 мс	10 мин 51 с 585 мс	79,43	154,89

Сондықтан Кесте 3 нәтижелері GPU негізіндегі кинетикалық солвердің тиімділігі реакциялар саны артқан сайын жоғарылайтынын көрсетеді. Сонымен қатар shared memory қолдану global memory-мен салыстырғанда есептеу уақытын едәуір қысқартып, үлкен өлшемді жану процестерінің кинетикалық теңдеулер жүйесін жоғары өнімді параллель есептеу жүйелерінде тиімді шешуге мүмкіндік береді.

3.6 Параллель кинетикалық солверді құру сипаттамасы

CUDA – бұл GPU (графикалық процессор) арқылы есептеу өнімділігін едәуір арттыра отырып, NVIDIA-ң параллелдік есептеуінің архитектурасы. Кинетикалық химия теңдеулерін есептеу барысында мұндай параллелизациялау технологиясын пайдалану уақыт пен есептеу кемшіліктерін азайтады [74].

GPU негізіндегі кинетикалық солверді басқаруға арналған бағдарламалық жүйенің жалпы архитектурасы 20-шы суретте көрсетілген. Жүйе бірнеше негізгі деңгейден тұрады: аппараттық орта, есептеу модульдері, деректер қоры, басқарушы және қолданбалы деңгей, сондай-ақ пайдаланушы интерфейстері. Мұндай құрылым есептеу процесін модульдік түрде ұйымдастыруға, деректерді орталықтандырылған түрде сақтауға және CPU–GPU арасындағы өзара әрекеттесуді тиімді басқаруға мүмкіндік береді.



Сурет 20 – CPU-GPU есептеу модульдері, деректер қоры және пайдаланушы интерфейстері біріктірілген бағдарламалық жүйенің архитектурасы

Аппараттық орта деңгейінде орталық процессор, жедел жад, графикалық адаптер және GPU жадысы орналасады. Орталық процессор жалпы басқару операцияларын, бастапқы деректерді дайындауды және есептеу нәтижелерін өңдеуді орындайды. Жедел жад CPU жағындағы аралық деректерді сақтауға қолданылады. Графикалық адаптер негізгі параллель есептеулерді орындауға жауап береді, ал device memory GPU ядролары пайдаланатын деректерді сақтауға арналған. CPU мен GPU арасындағы деректер алмасу PCIe шинасы арқылы жүзеге асырылады.

Есептеу модульдері деңгейі екі негізгі бөліктен тұрады. Бірінші бөлігі — C++ native модулі, ол CPU-side preprocessing және postprocessing кезеңдерін орындайды. Бұл модуль бастапқы параметрлерді дайындау, деректерді есептеуге қолайлы форматқа келтіру және алынған нәтижелерді өңдеу үшін қолданылады. Екінші бөлігі — CUDA модулі, онда параллель есептеу ядролары іске қосылады. CUDA модулі kernel launch операцияларын, host–device жады алмасуын және GPU-да орындалатын негізгі есептеу процедураларын қамтамасыз етеді.

Деректер қоры деңгейінде SQL Server қолданылады. Деректер қорында есептеу параметрлері, пайдаланушылар туралы мәліметтер, есептеу нәтижелері және жүйелік журналдар сақталады. Бұл тәсіл есептеу нәтижелерін кейінгі талдау үшін жинақтауға, пайдаланушы сұраныстарын бақылауға және әртүрлі есептеу сценарийлерін салыстыруға мүмкіндік береді.

Басқарушы және қолданбалы деңгей C#/NET технологиясы негізінде ұйымдастырылады. Бұл деңгейде басқарушы модуль, service layer, контроллерлер және логикалық өңдеу компоненттері орналасады. Басқарушы модуль есептеу процесін іске қосу, сұраныстарды қабылдау, деректерді тексеру, есептеу модульдерімен байланыс орнату және нәтижелерді пайдаланушыға

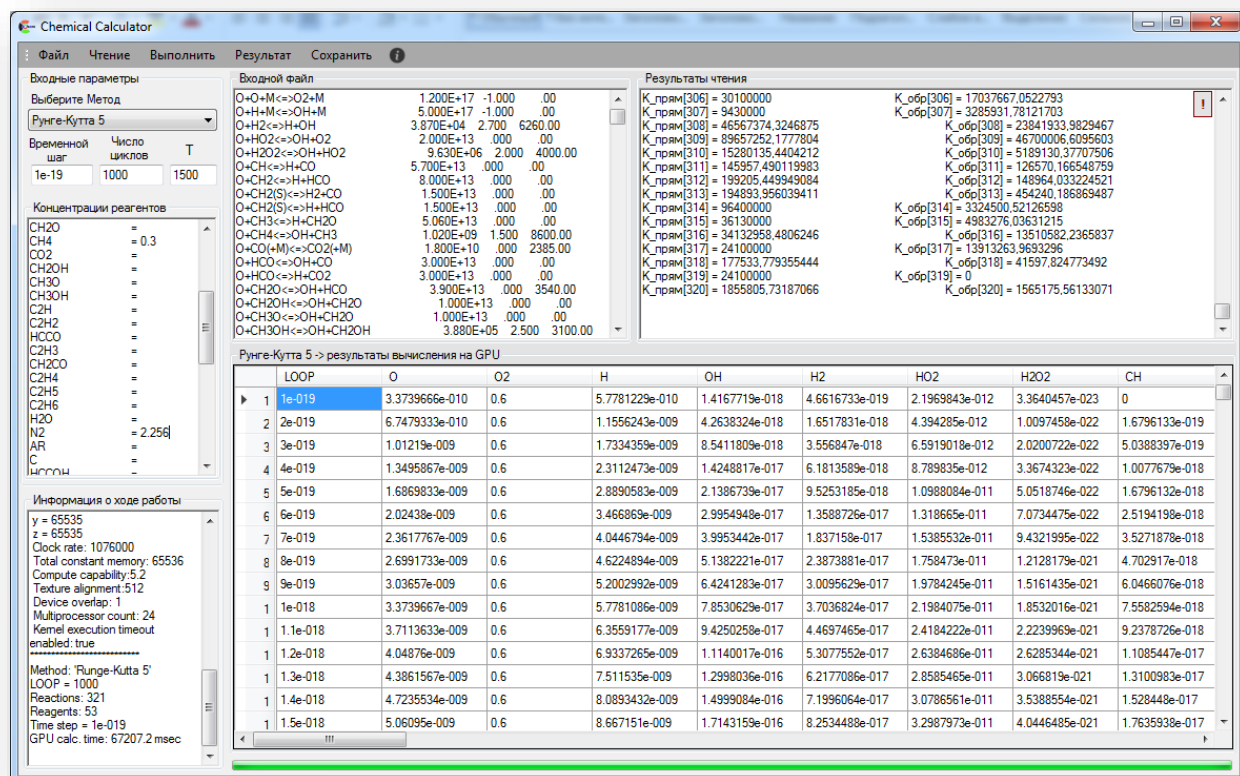
қайтару қызметтерін атқарады. Контроллерлер мен логикалық модульдер пайдаланушы сұраныстарын өңдеп, оларды есептеу модульдеріне немесе деректер қорына бағыттайды.

Пайдаланушы интерфейстері екі түрлі клиент арқылы жүзеге асырылады: web клиент және desktop клиент. Web клиент ретінде ASP.NET MVC негізіндегі интерфейс қарастырылған, ал desktop клиент Windows Forms технологиясы арқылы жасалады. Бұл қолданушыларға есептеу параметрлерін енгізуге, есептеу процесін іске қосуға, нәтижелерді көруге және сақталған мәліметтермен жұмыс істеуге мүмкіндік береді.

Осы архитектураның маңызды ерекшелігі — есептеу логикасының басқарушы деңгейден бөлек орналасуы. CPU және GPU есептеу модульдері жеке компоненттер ретінде ұйымдастырылғандықтан, жүйені кеңейту, оңтайландыру және әртүрлі есептеу әдістерін қосу жеңілдейді. Сонымен қатар деректер қорының жеке деңгей ретінде қолданылуы есептеу нәтижелерін жүйелі түрде сақтап, оларды кейінгі ғылыми талдау үшін пайдалануға мүмкіндік береді.

Жалпы алғанда, ұсынылған архитектура жоғары өнімді GPU есептеулерін қолданбалы бағдарламалық жүйемен біріктіруге бағытталған. Бұл құрылым жану процестерінің кинетикалық теңдеулерін есептеу кезінде CUDA негізіндегі параллель алгоритмдерді тиімді іске асыруға, есептеу нәтижелерін сақтауға және оларды пайдаланушы интерфейстері арқылы ыңғайлы түрде басқаруға мүмкіндік береді.

Параллель кинетикалық солверді жүзеге асыру барысында химиялық реакцияға түсетін жүйелердің кинетикасын нөлдік жуықтауда модельдеу үшін «Химиялық есептеуші» бағдарламасы жасалды. Бағдарламадағы жұмыс «CUDA» технологиясын пайдалана отырып есептеу жүйесімен қатар, орталық процессор мен NVIDIA-ң графикалық процессорының жұмысын қоса қамтиды. Бағдарламаның графикалық интерфейсі C# программалау тілінде жазылған [13, 14]. Бағдарламаның интерфейсі химиялық реактивті жүйелерді шешу барысында жоғары деңгейде ыңғайлы болу үшін тиімді жазылған. Бағдарламалық жасақтаманың көрінісі 21-суретте бейнеленген.



Сурет 21 – «Химиялық есептеуіш» бағдарламалық жасақтаманың интерфейсі

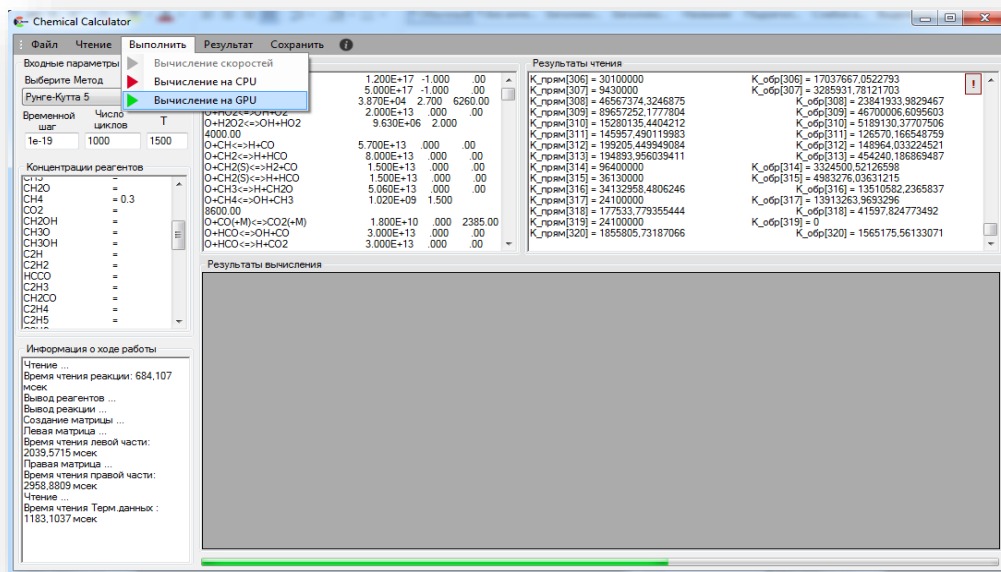
Әзірленген бағдарламада есептеу жұмыстарын жүргізу үшін Chemkin механизмiнiң қысқартылған файлы қолданылады. Бұл тәсіл пайдаланушыға өзінің көлемді кинетикалық механизмдерін Chemkin форматына ыңғайлы түрде түрлендіріп, бағдарламаға енгізуге мүмкіндік береді. Пайдаланушы қажетті мәліметтерді мәзірдегі арнайы бөлімдер арқылы көшіре алады: Файл → Реакциялар және Файл → Термодинамикалық қасиеттер.

Көшіру үдерісі аяқталғаннан кейін енгізілген мәліметтер «Кіретін файл» қосымшасында көрсетіледі. Бұл бөлімде реакциялар мен заттардың термодинамикалық қасиеттерін бір-бірінен ажырататын «THERM» сөзі жазылған арнайы жол орналасады. «Кіретін файл» қосымшасындағы мәліметтер жол бойынша оқылады, ал оқу нәтижелері «Оқу қорытындылары» қосымшасында сақталады.

Файлды оқу барысында берілген механизмдегі реагенттер мен реакциялар саны, Аррениус тендеуінің коэффициенттері, реакциялардың бағыты, реакциялық механизм үшін сол және оң жақ матрицалардың шешімдері, сондай-ақ механизм құрамындағы әрбір элементтің термодинамикалық мәліметтері анықталады. Бұл мәліметтерді оқу үдерісі мәзірдегі Оқу → Файлдағы деректерді оқу командасын таңдау арқылы жүзеге асырылады.

Есептеу нәтижелерін алу үшін пайдаланушы ашылмалы тізімнен қажетті есептеу тәсілін таңдап, бастапқы кіріс мәліметтерін енгізеді. Бағдарламада есептеулерді орталық процессорда да, графикалық процессорда да орындау мүмкіндігі қарастырылған. Орталық процессормен салыстырғанда NVIDIA

графикалық процессорын қолдану реакциялар саны артқан жағдайда есептеу уақытын едәуір қысқартуға мүмкіндік береді. Бұл нәтиже 22-суретте көрсетілген.



Сурет 22 – CPU немесе GPU бөлімдеріндегі есептеулер

Химиялық реакциялар жүйесін кинетикалық тұрғыдан модельдеу үшін 4-ші және 5-ші ретті Рунге–Кутта әдістері қолданылды. Осы әдістер негізінде орталық процессорда тізбекті есептеулерді орындауға арналған C++ бағдарламалық коды әзірленді. Ал параллель есептеулерді жүзеге асыру үшін CUDA параллельді есептеу технологиясы пайдаланылды.[75]

4-ші және 5-ші ретті Рунге–Кутта әдістері арқылы концентрация өзгерісін есептеу алгоритмі бірнеше негізгі кезеңнен тұрады. Алдымен массивтер үшін қажетті жады бөлінеді. Одан кейін файлдан кіріс мәліметтері оқылады. Бұл мәліметтерге тура және кері реакциялардың жылдамдықтары, бастапқы концентрациялар, сондай-ақ реакциялық механизмнің оң және сол жақ матрицалары жатады. Келесі кезеңде берілген уақыт аралығында реакциялық механизм құрамындағы әрбір компоненттің концентрациясының өзгерісі есептеледі. Есептелген нәтижелер бойынша концентрациялардың жаңа мәндері жазылады. Соңында уақыт бойынша шарт тексеріледі: егер ағымдағы уақыт қадамы жалпы есептеу уақытының шегінен аспаса, есептеу үдерісі қайта жалғасады, ал кері жағдайда есептеу тоқтатылады.

Графикалық процессорда есептеу жүргізу кезінде 4-ші және 5-ші ретті Рунге–Кутта әдістері бойынша деректерді параллельдеу алгоритмі қолданылады. Алдымен файлдағы кіріс мәліметтері, яғни тура және кері реакциялардың жылдамдықтары, концентрациялар және реакциялық механизмнің оң және сол жақ матрицалары оқылады. Содан кейін бұл мәліметтерді графикалық құрылғыға көшіру үшін жады бөлінеді. Кіріс деректері жедел жадыдан графикалық процессордың жадына тасымалданады. Бұдан кейін

блоктағы ағындар саны және тордағы блоктар саны анықталады.

Берілген уақыт аралығында реакциялық механизмнің әрбір компоненті үшін концентрация өзгерісін есептеу GPU ядролары арқылы орындалады. Бұл кезеңде алдымен графикалық процессорда Рунге–Кутта әдісінің бірінші коэффициентін есептеу функциясы шақырылады. Кейін екінші және үшінші коэффициенттерді есептеу функциялары орындалып, берілген уақыт қадамындағы концентрация өзгерісі анықталады. Егер 4-ші ретті Рунге–Кутта әдісі қолданылса, төртінші коэффициентті есептеу функциясы шақырылады. Ал 5-ші ретті Рунге–Кутта әдісі үшін бесінші коэффициентті есептеу функциясы қосымша орындалады. Осыдан кейін барлық заттар концентрацияларының мәндері жаңа есептелген мәндермен алмастырылады. Соңғы кезеңде GPU жадындағы нәтижелер CPU жадына көшіріліп, концентрациялардың жаңа мәндері сақталады.

Есептеу үдерісінде уақыт шарты тексеріліп отырады. Егер ағымдағы уақыт қадамы жалпы есептеу уақытына жетпесе, концентрацияларды есептеу кезеңі қайта орындалады. Ал берілген уақыт толық аяқталған жағдайда есептеу тоқтатылады.

CPU және GPU есептеу уақыттарын салыстырмалы талдау нәтижелері төмендегі кестеде көрсетілген. GPU-да деректерді параллельдеу реакцияларды жеке ағындарға бөлу арқылы жүзеге асырылады. Әрбір ағын берілген уақыт аралығында белгілі бір зат концентрациясының өзгерісін есептейді. Параллельдеу реакциялар деңгейінде орындалатындықтан, механизмдегі реакциялар саны артқан сайын GPU-да есептеу тиімділігі жоғарылайды және есептеу уақытын қысқартуға мүмкіндік береді.

Әзірленген бағдарламада есептеу жүргізу үшін Chemkin механизмінің қысқартылған файлы қолданылады. Бұл пайдаланушыға өзінің көлемді кинетикалық механизмдерін Chemkin форматына ыңғайлы түрде түрлендіріп, бағдарламаға енгізуге мүмкіндік береді. Есептеу нәтижелерін алу үшін пайдаланушы ашылмалы тізімнен қажетті әдісті тандап, бастапқы кіріс мәліметтерін енгізеді. Бағдарламада есептеулерді орталық процессорда да, графикалық процессорда да орындау мүмкіндігі қарастырылған. Орталық процессормен салыстырғанда NVIDIA графикалық процессорын қолдану реакциялар саны көп болған жағдайда есептеу уақытын едәуір қысқартуға мүмкіндік береді.

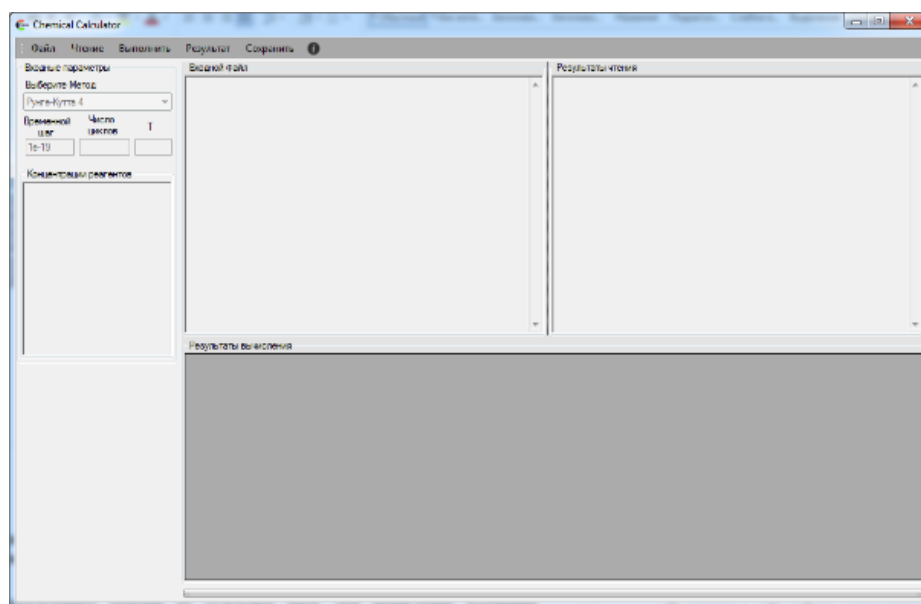
Бағдарлама Chemkin бағдарламалық кешені форматындағы кіріс файлдарынан алынған деректерді автоматты түрде оқуға, реакция жылдамдықтарын есептеуге және уақыт бойынша заттар концентрациясының өзгерісін анықтауға арналған. Негізгі есептеулер 4-ші және 5-ші ретті Рунге–Кутта әдістері арқылы орындалады. Орталық процессордағы есептеулер тізбекті алгоритм бойынша жүзеге асырылады, ал NVIDIA графикалық картасындағы есептеулер CUDA технологиясы негізінде параллель түрде жүргізіледі.

Бағдарламаны іске қосқан кезде алдымен бастапқы интерфейс ашылады. Бұл интерфейс бірнеше негізгі компоненттен тұрады. Терезенің жоғарғы бөлігінде мәзір жолағы орналасқан. Бағдарламаға деректерді енгізу Файл мәзірі

арқылы орындалады. Реакциялық механизмді көшіру үшін Файл → Реакциялар командасы, ал термодинамикалық деректерді енгізу үшін Файл → Термодинамикалық қасиеттер командасы таңдалады. Көшірілген деректер бағдарламаның «Входной файл» аймағына импортталады. Бұл аймақта реакциялар мен термодинамикалық қасиеттерге қатысты кіріс мәліметтері «THERMDAT» сөзі арқылы бөлінеді. Кіріс деректерін оқу кезінде термодинамикалық мәліметтер осы белгі арқылы анықталады. Қажет болған жағдайда механизм құрамындағы кіріс деректерін тікелей бағдарлама ішінде өзгертуге болады.

Кіріс деректерін оқу мәзірдегі Оқу → Файлдағы деректерді оқу командасын таңдау арқылы орындалады, бұл үдеріс 23-суретте көрсетілген. Аталған команда іске қосылғаннан кейін бағдарламада алдын ала берілген алгоритм бойынша деректерді автоматты түрде оқу басталады.

Реакциялық механизмді оқу барысында есептеулерге қажетті негізгі мәліметтер анықталады. Атап айтқанда, әрбір реакция үшін тура және кері бағыттағы жылдамдықтар есептеледі, сондай-ақ реакция жылдамдығын сипаттайтын қажетті параметрлер қалыптастырылады. Нәтижесінде бағдарлама келесі есептеу кезеңдерінде қолданылатын реакция жылдамдықтарының мәндерін автоматты түрде дайындайды.

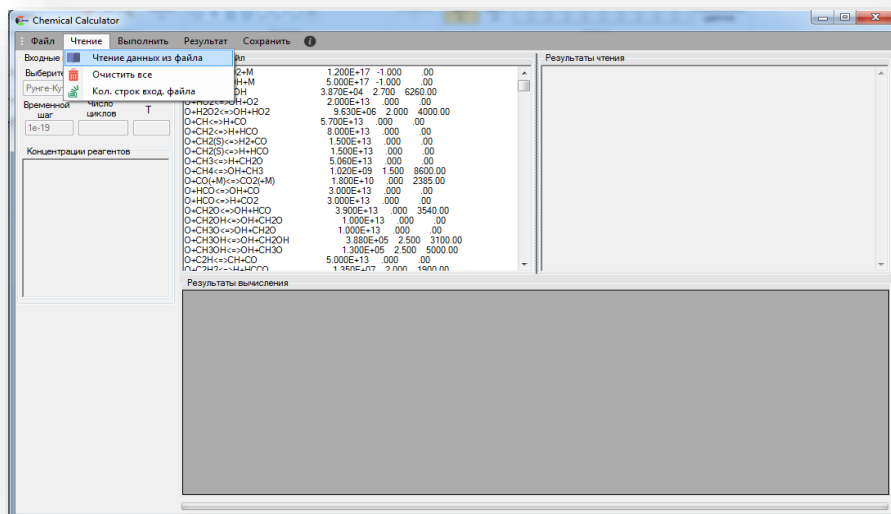


Сурет 23 – Бағдарламаның басты интерфейсі

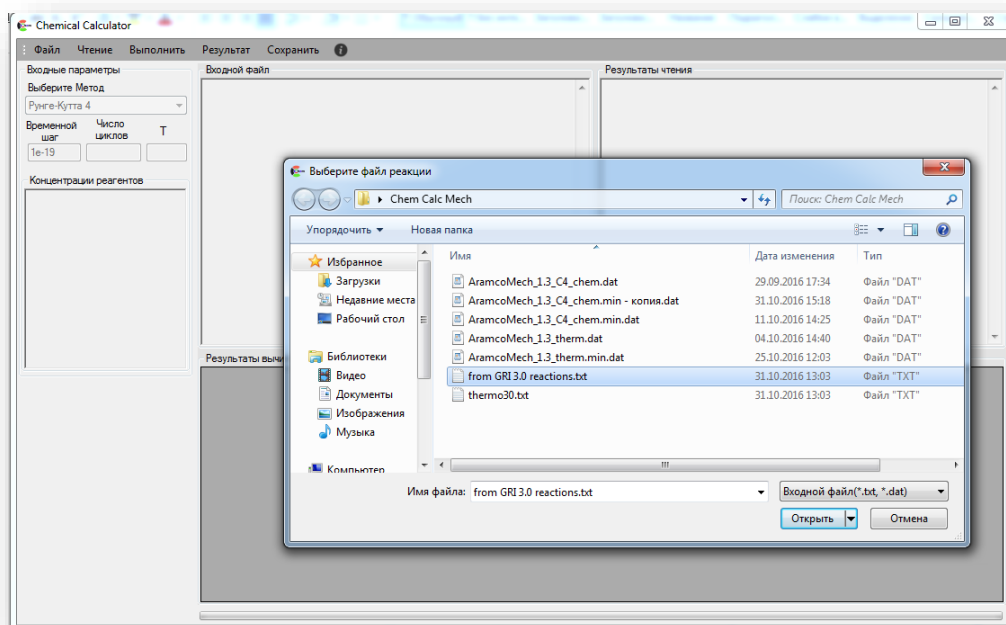
Бірінші параметр — предэкспоненциальный коэффициент, второй параметр — температурный коэффициент, третий параметр — энергия активации, а четвертый параметр — направление реакции, указывающее на то, в каком направлении протекает реакция. Эти параметры используются для расчета скорости реакции по уравнению Аррениуса.

Если реакция протекает в одном направлении, то ее скорость рассчитывается только для этого направления. Если реакция обратима, то ее скорость рассчитывается для обоих направлений. В этом случае программа выдает две скорости: одну для прямого, другую для обратного процесса.

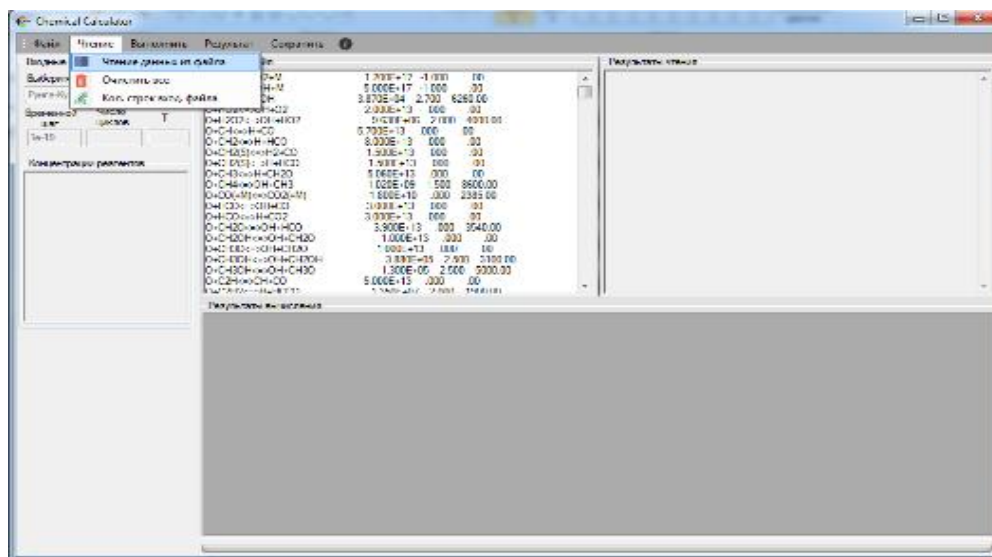
механизм оқылғаннан кейін бағдарлама механизм құрамына кіретін барлық компоненттерді анықтап, оларды кейінгі есептеу кезеңдерінде қолдануға дайындайды.



Сурет 24 – Деректерді көшіруге арналған мәзірдің көрінісі



Сурет 25 – Кіретін деректерді көшіруге арналған терезе



Сурет 26 – Көшірілген деректерді оқу

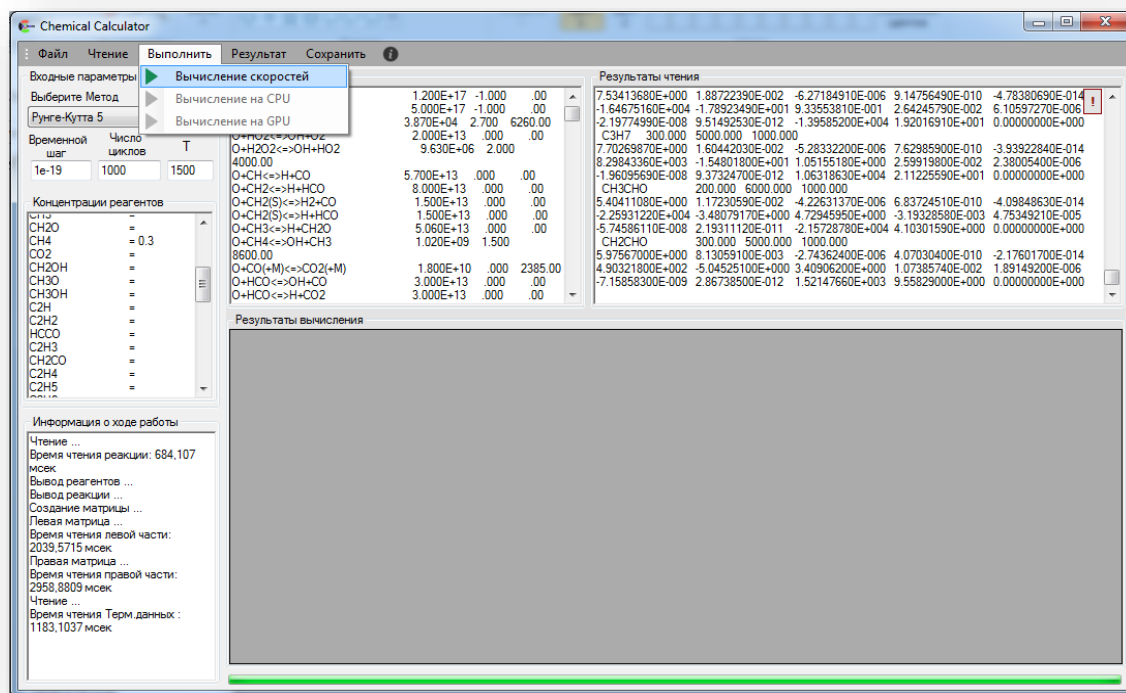
Осыдан кейін деректер матрицасының оң және сол жақ бөліктері анықталады. Бұл матрицаларда берілген реакцияға қатысатын компоненттердің стехиометриялық коэффициенттері сақталады. Аталған мәліметтер кейін реакция жылдамдығын есептеу кезінде қолданылады.

Механизм құрамындағы әрбір реагенттің термодинамикалық қасиеттері кіріс файлындағы термодинамикалық деректер негізінде есептеледі. Осы есептелген термодинамикалық сипаттамалар арқылы заттың жылу сыйымдылығы, энтальпиясы, энтропиясы, Гиббс еркін энергиясы және Гельмгольц еркін энергиясы анықталады.

«Кіріс деректері» аймағында реакция жылдамдықтарын есептеу үшін алдымен қоспаның бастапқы температурасы енгізіледі. Содан кейін мәзірдегі Іске қосу → Жылдамдықтарды есептеу командасы таңдалады. Осы әрекет орындалғаннан кейін бағдарлама әрбір реакция үшін тура және кері бағыттағы реакция жылдамдықтарын есептейді. Бұл үдеріс 26-суретте көрсетілген.

Әзірленген бағдарламада негізгі есептеу жұмыстарын екі түрлі тәсілмен орындауға болады: орталық процессордағы тізбекті алгоритм арқылы және графикалық процессордағы параллель алгоритм арқылы. Есептеуді бастамас бұрын пайдаланушы концентрацияның уақыт бойынша өзгеруін есептеу әдісін таңдайды. Бұдан кейін уақыт қадамы мен есептеу циклдерінің саны беріледі.

Кіріс мәліметтері негізінде концентрация өзгерісін есептеу үшін қажетті есептеу ресурстары пайдаланылады. Егер реакциялық механизмнің көлемі үлкен болса, онда тізбекті алгоритмді қолдану көп есептеу шығынын және едәуір уақытты талап етеді. Ал параллель алгоритмді қолдану реакциялар саны артқан жағдайда есептеуді жылдамдатуға мүмкіндік береді. Сондықтан графикалық процессорда орындалатын параллель есептеу тәсілі ауқымды кинетикалық механизмдер үшін тиімді болып табылады.

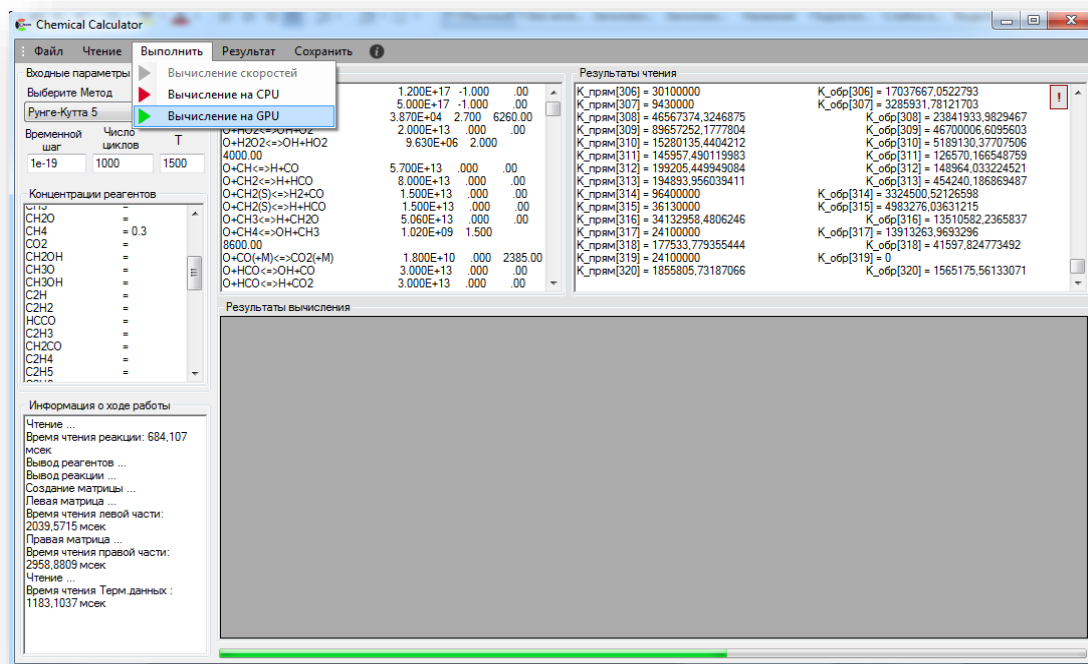


Сурет 27 – Реакцияның жылдамдықтарын есептеу

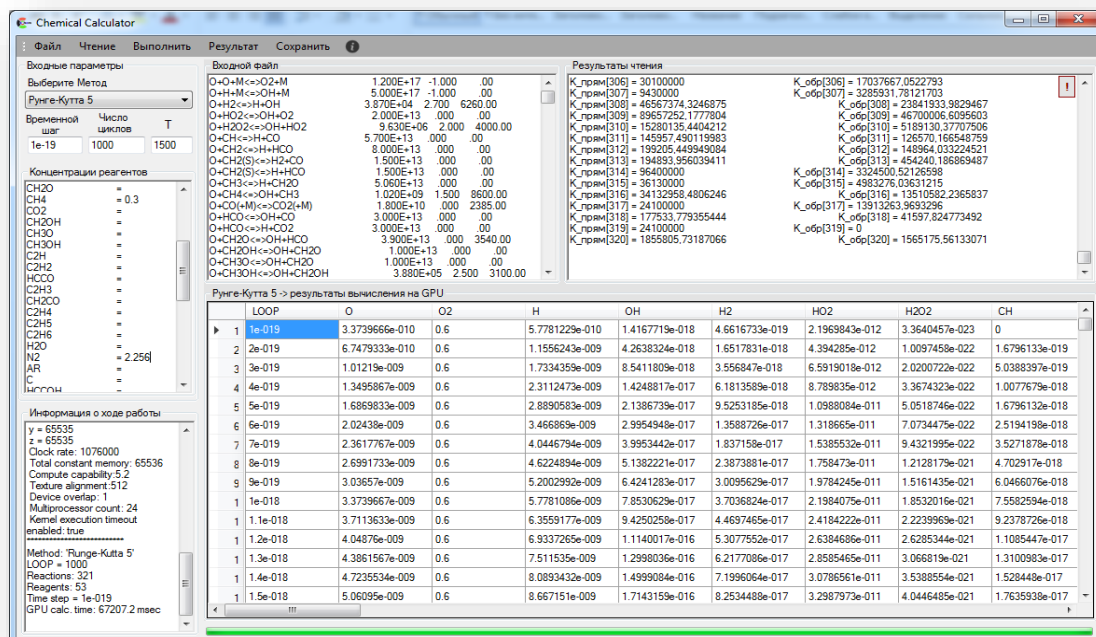
Ары қарай «Заттың концентрациясы» аймағында есептеулерді жүргізу үшін механизм құрамындағы реагенттердің бастапқы концентрациялары енгізіледі. Егер кейбір заттар үшін бастапқы концентрация мәндері берілмесе, онда бағдарлама бұл реагенттердің бастапқы концентрациясын нөлге тең деп қабылдайды. Яғни мұндай жағдайда $C_0 = 0$ шарты қолданылады. Бұл үдеріс 27-суретте көрсетілген.

Есептеу нәтижелері бағдарламада кесте түрінде беріледі, бұл 29-суретте көрсетілген. Нәтижелерді көрнекі түрде талдау үшін оларды график түрінде көрсету мүмкіндігі қарастырылған. Ол үшін мәзірден Қорытынды → Қорытындыны GPU/CPU-да көрсету → Графикте көрсету командаларын таңдау қажет.

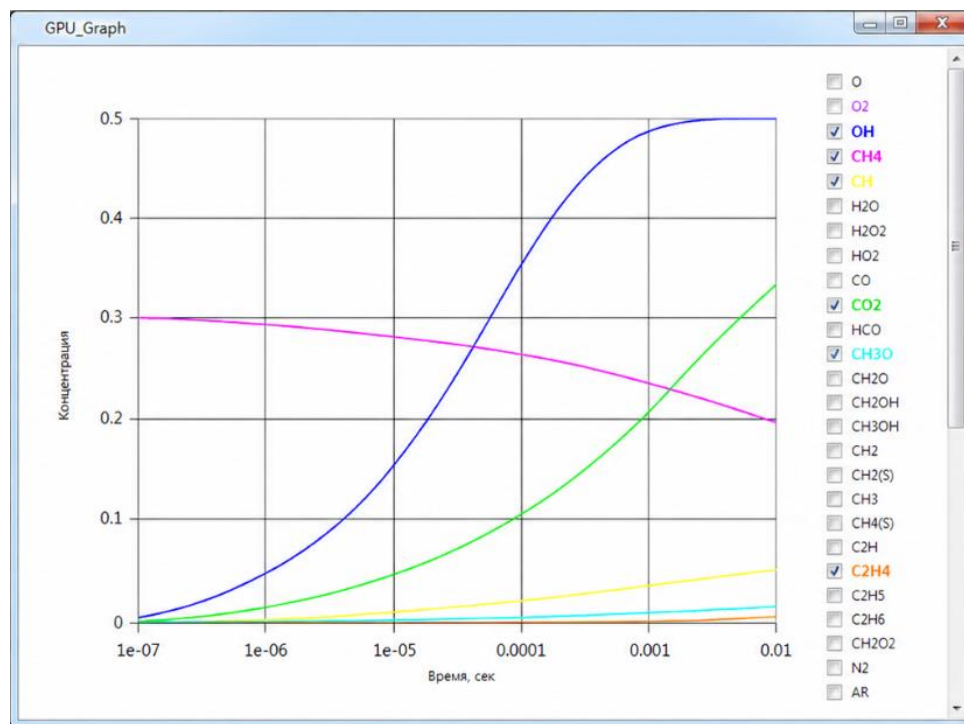
Осы әрекет орындалғаннан кейін график салынатын аймақ және оның оң жағында механизм құрамындағы реагенттердің атаулары мен белгілеулері көрсетілген терезе ашылады. Пайдаланушы қажетті реагенттің жанына құсбелгі қою арқылы сол зат концентрациясының уақыт бойынша өзгеру графигін шығара алады. Ал құсбелгіні алып тастаған жағдайда, тиісті реагенттің графигі экраннан жойылады. Осылайша бағдарлама әрбір компоненттің концентрация өзгерісін жеке немесе бірнеше реагент бойынша салыстырмалы түрде талдауға мүмкіндік береді.



Сурет 28 – Есептеуді графикалық процессорда жүргізу



Сурет 29 – Есептеудің нәтижелері



Сурет 30 – Уақыт бойынша заттың концентрациясының өзгерісі

3.7 Бағдарламалық архитектураны ұйымдастыру

Физикалық сервер бір уақытта веб-сервер және есептеу сервері қызметтерін атқаратындықтан, веб-қолжетімділікті қамтамасыз ету және есептеу жұмыстарын пайдаланушылардың жергілікті компьютерлеріне артық жүктеме түсірмей орындау мақсатында IIS, SQL Server және CUDA технологиялары негізінде серверлік бағдарламалық архитектура ұйымдастырылды. Мұндай архитектурада IIS, ASP.NET MVC және SQL Server технологияларын біріктіру пайдаланушы сұраныстарын өңдеуге, есептеу нәтижелерін сақтауға және GPU ресурстарын сервер арқылы қашықтан пайдалануға мүмкіндік береді.

Қызметтерді, қосымшаларды және веб-тораптарды сенімді орналастыру үшін Windows Server операциялық жүйесі қолданылды. Бұл операциялық жүйе қауіпсіз, басқаруға ыңғайлы және кеңейтілетін модульдік платформаны қамтамасыз етеді. Осыған байланысты веб-сервис архитектурасын ұйымдастыру үшін IIS веб-сервері таңдалды. IIS қолдану пайдаланушылардың ақпаратқа және есептеу сервисіне жергілікті желі арқылы да, ауқымды желі, яғни Интернет арқылы да қол жеткізуін қамтамасыз етеді.

Пайдаланушы интерфейсін автоматтандыру интерфейс элементтерімен автоматты тестілеу скрипттері арқылы өзара әрекеттесуге мүмкіндік береді.

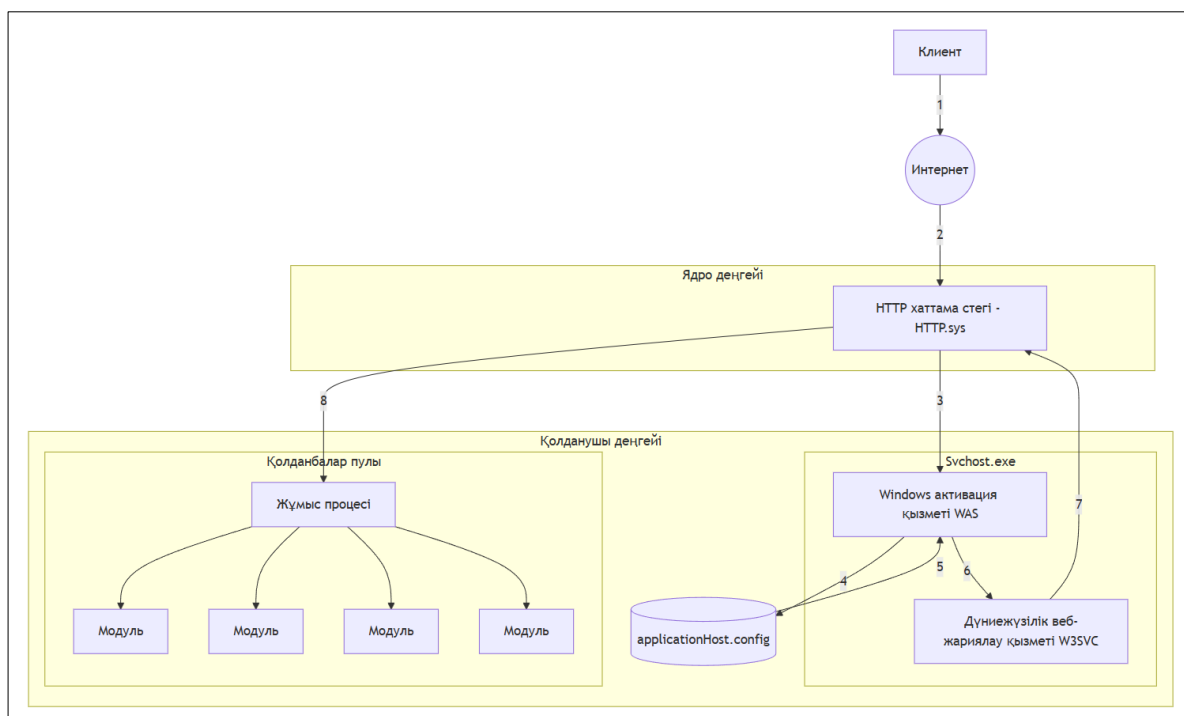
Аталған технологияларды қолдану келесі артықшылықтарды қамтамасыз етеді:

- қосымшаларды автоматты түрде оқшаулау және сервер ресурстарын тиімді пайдалану арқылы веб-қауіпсіздік деңгейін арттыру;
- қауіпсіздік тәуекелдерін төмендету мақсатында жұмыс процестерін бірегей сәйкестендіру және оларды оқшауланған ортада іске қосу;
- пайдаланушы қажеттіліктеріне сәйкес IIS модульдік компоненттерін оңай қосу, өшіру немесе алмастыру мүмкіндігі;

- кіріктірілген динамикалық кэштеу және кеңейтілген қысу механизмдері арқылы веб-қосымшалардың жұмыс жылдамдығын арттыру;
- веб-қосымшаны басқару және IIS компоненттерін баптау үшін IIS қызметтер диспетчерін қолдану;
- веб-қосымша иелеріне файлдарды жіберу және жүктеу мүмкіндігін беру үшін FTP хаттамасын пайдалану;
- сервердегі бір веб-сайттың басқа веб-сайттардың жұмысына әсерін болдырмау үшін веб-қосымшаларды оқшаулау;
- Classic ASP, ASP.NET сияқты әртүрлі технологиялар негізінде құрылған веб-қосымшаларды баптау;
- веб-серверді басқаруға қатысты негізгі міндеттердің үлкен бөлігін PowerShell арқылы орындау;
- IIS көмегімен бірнеше веб-серверді бір серверлер фермасына біріктіру;
- NUMA архитектурасының артықшылықтарын пайдалану және сервердің тиімді өнімділігін қамтамасыз ету.

Аталған артықшылықтар есептеуіш ядросы бар веб-сервисті орналастыруға мүмкіндік беретін серверлік қызметтерді ұйымдастыру архитектурасы негізінде жүзеге асырылады.

Серверлік бөлімде деректерді өңдеу келесі кезеңдер бойынша жүзеге асырылады, бұл үдеріс 31-суретте көрсетілген.



Сурет 31 – Веб-сервердің жүзеге асу схемасы

Бір уақытта өңделетін сұраныстар санын арттыру үшін жүйеде енгізу-шығару амалдарының аяқталуын өңдейтін арнайы API қолданылады. Бұл технология Input/Output Completion Port, яғни I/O Completion Port деп аталады. Сонымен қатар IIS құрамында сұраныстар кезегін басқаруға және енгізу-шығару амалдарын өңдеуге арналған ағындар жиыны қарастырылған. Сұраныстарды

өңдеу барысында аталған API HTTP.sys драйверімен өзара байланыста жұмыс істейді.

HTTP.sys драйверінің адаптері болып табылатын World Wide Web Publishing Service, яғни W3SVC қызметі келесі негізгі міндеттерді атқарады:

- HTTP.sys драйверінің конфигурациясын қалыптастыру;
- өнімділік көрсеткіштеріне мониторинг жүргізу;
- жұмыс процестерін басқару үшін Windows Process Activation Service, яғни WAS қызметімен өзара әрекеттесу.

WAS қызметінің негізгі компоненттеріне конфигурация менеджері, процестер менеджері және басқарылмайтын тыңдаушы адаптерлер интерфейсі жатады. Конфигурация менеджері applicationHost.config файлынан қосымшалар жиыны мен веб-қосымшалар туралы конфигурациялық мәліметтерді оқиды. Процестер менеджері қосымшалар жиыны мен жұмыс процестері арасындағы сәйкестікті қамтамасыз етеді. Егер белгілі бір қосымшалар жиынына сәйкес жұмыс процесі іске қосылмаған болса, процестер менеджері оны автоматты түрде іске қосады. Ал басқарылмайтын тыңдаушы адаптерлер интерфейсі HTTP хаттамасына жатпайтын сұраныстарды өңдейтін тыңдаушылар үшін қажетті интерфейсін қамтамасыз етеді.

Серверде клиент тарапынан келетін пайдаланушы сұраныстарын өңдеу архитектурасы бірнеше кезеңнен тұрады. Алдымен пайдаланушы браузер арқылы сервердегі белгілі бір ресурсқа HTTP-сұраныс жібереді. Бұл сұранысты HTTP.sys драйвері қабылдайды. Сұралған URL мекенжайына қатысты конфигурациялық деректерді алу үшін HTTP.sys драйвері WAS қызметіне жүгінеді.

Одан кейін WAS құрамындағы конфигурация менеджері applicationHost.config файлынан қажетті мәліметтерді оқиды. Бұл мәліметтерге, әдетте, сайттың конфигурациялық баптаулары және қосымшалар жиыны туралы ақпарат кіреді. WAS оқылған деректерді World Wide Web Publishing Service, яғни W3SVC қызметіне береді.

W3SVC қызметі WAS арқылы алынған деректерді HTTP.sys драйверінің конфигурациясын баптау үшін пайдаланады. HTTP.sys драйвері келіп түскен сұраныстарды WAS өңдейтін енгізу-шығару аяқталу порттарының, яғни I/O Completion Port кезегіне орналастырады.

WAS сұраныстар кезегін өңдеу үшін ағындар жиынын қолданады. Егер сол сәтте сұралған URL мекенжайына қатысты жұмыс процесі іске қосылмаған болса, онда ол автоматты түрде іске қосылады. Бұл жағдайда ағындар жиыны компьютердің бір ядросына 250-ге дейін ағынды пайдалана алады.

Іске қосылған жұмыс процесі ішінде ағымдағы уақытта қанша сұраныс өңделіп жатқанын тексеру орындалады. Егер өңделіп жатқан сұраныстар саны қалыпты шектен, яғни 5000 сұраныстан асып кетсе, жаңа сұраныс кезекке қойылады.

Егер сұраныс статикалық файлға бағытталған болса, онда IIS файл мазмұнын енгізу-шығару аяқталу порты арқылы сұраныс жіберген пайдаланушыға қайтарады. Ал қалған жағдайларда сұраныс ASP.NET CLR

ағындар жиынына беріледі.

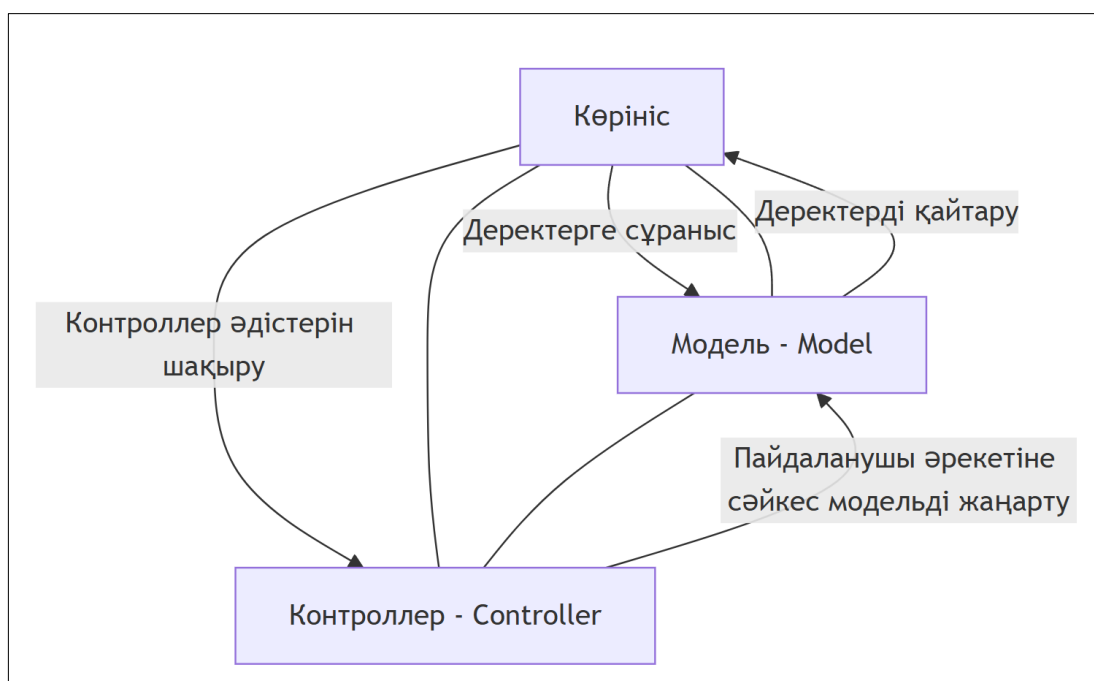
CLR ортасы сұранысты өңдейді. Осы кезеңде веб-серверде орналасқан веб-қосымшаның бағдарламалық коды орындалады. CLR ортасында сұранысты өңдеу аяқталғаннан кейін нәтиже HTTP.sys драйверіне жіберіледі, ал ол өз кезегінде нәтижені IIS енгізу-шығару аяқталу порты арқылы пайдаланушыға қайтарады. Нәтижесінде пайдаланушы өңделген жауапты алады.

Пайдаланушылардың сұраныстарын веб-қосымша деңгейінде өңдеу және есептеулерді графикалық адаптерде орындауға қажетті деректерді үлестіру үшін MVC архитектуралық үлгісі және объектіге бағытталған C# бағдарламалау тілі қолданылды. MVC үлгісі Model, View және Controller компоненттерінен тұрады, бұл құрылым 32-суретте көрсетілген.

MVC архитектуралық үлгісінің негізгі үш компоненті бар. Model, яғни модель, қосымшаның негізгі деректер құрылымын және бизнес-логикасын сипаттайды. Ол пайдаланушы интерфейсінен тәуелсіз түрде пәндік аймақтың мәліметтерін және олармен орындалатын негізгі амалдарды қамтиды.

View, яғни көрсетілім, белгілі бір контекстегі модель деректерінің визуалды бейнесін қалыптастырады. Әдетте көрсетілім пайдаланушыға жіберілетін нәтижелік интерфейс немесе веб-бет түрінде ұсынылады.

Controller, яғни контроллер, модель мен көрсетілім арасындағы байланысты қамтамасыз ететін басқарушы компонент болып табылады. Ол пайдаланушыдан келетін кіріс деректерін қабылдап, оларды өңдейді және қандай әрекет орындалуы қажет екенін анықтайды. Мысалы, контроллер белгілі бір көрсетілімді қайтаруы немесе сұранысты веб-сервиске жіберуі мүмкін.



Сурет 32 - MVC (Model, View, Controller) деректерді жіберу схемасы

Веб-ресурс арқылы деректерді жіберу және есептеу жүйесіне қолжетімділікті жобалау барысында ASP.NET MVC технологиясы қолданылды.

Бұл технология осы типтегі жүйелерді құруда бірқатар маңызды артықшылықтарға ие. ASP.NET MVC веб-хаттамалар негізінде жұмыс істейтін, деректерді тиімді және ыңғайлы түрде тасымалдауға арналған ауқымды жүйелерді жүзеге асыруға мүмкіндік береді. Сонымен қатар ол дәстүрлі веб-қосымшаларды әзірлеу кезінде кездесетін бірқатар шектеулерді жоюға бағытталған.

ASP.NET MVC технологиясының негізгі артықшылықтарының бірі — MVC архитектуралық үлгісін қолдану және әрбір HTTP сұранысын нақты бір әдіске бағыттау мүмкіндігі. Бұл HTTP деңгейіндегі артық абстракцияларды азайтып, сұраныстарды өңдеу логикасын нақты әрі түсінікті ұйымдастыруға мүмкіндік береді.

Аталған архитектурада пайдаланушы интерфейсі басқарушы логикадан, яғни контроллерден оқшауланады. Бұл мүмкіндік дұрыс ұйымдастырылған жағдайда контроллерге өзгерістер енгізу кезінде пайдаланушы интерфейсін өзгертпеуге жағдай жасайды. Нәтижесінде қосымшаны сүйемелдеу және дамыту үдерісі жеңілдейді.

Қосымша логикасының пайдаланушы интерфейсінен бөлінуі ASP.NET MVC негізінде құрылған жүйенің жеке компоненттерін тестілеуді жеңілдетеді. Мысалы, контроллер кластарын пайдаланушы интерфейсін тікелей тестілемей-ақ тексеруге болады. Сонымен қатар контроллерлер тестілеуге қолайсыз HttpContext класына тәуелсіз болғандықтан, автоматтандырылған тестілеу скрипттерін құру үдерісі жеңілдейді.

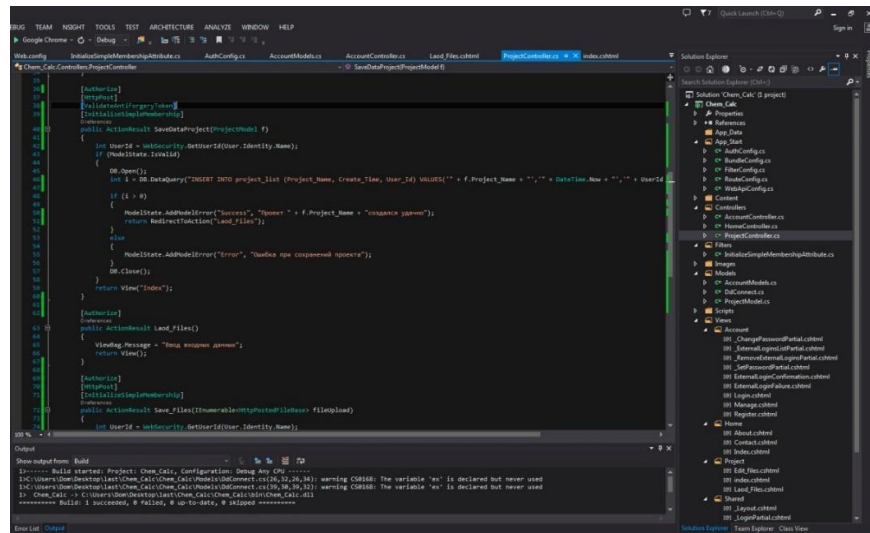
MVC үлгісінде әрекеттерге арналған сүзгілерді қолдану мүмкіндігі қарастырылған. Бұл механизм контроллер әрекеттерінің орындалу кезеңдерінде қосымша тексерулер, өңдеулер немесе шектеулер енгізуге мүмкіндік береді. Осылайша қосымшада орындалатын әрекеттерді басқару және бақылау тиімді түрде ұйымдастырылады.

ASP.NET MVC технологиясымен жұмыс істеу барысында жүйенің белгілі бір бөліктерін берілген уақыт аралығында кәштеу мүмкіндігі де қарастырылған. Бұл деректерге жылдам қол жеткізуге және веб-қосымшаның жалпы өнімділігін арттыруға мүмкіндік береді.

MVC архитектуралық үлгісінің тағы бір маңызды артықшылығы — Ajax, jQuery және басқа да JavaScript кітапханаларымен тиімді өзара әрекеттесу мүмкіндігі. Сонымен қатар ASP.NET MVC құрамында JSON форматындағы деректерді әдіс параметрлеріне түрлендіруді қолдау мүмкіндігі бар. Бұл клиенттік және серверлік бөліктер арасындағы деректер алмасуды жеңілдетеді.

Жоғарыда аталған технологиялық мүмкіндіктер деректерді өңдеу үдерісін жетілдіруге, жүйенің икемділігін арттыруға және оны клиенттік тараптың қажеттіліктеріне бейімдеуге мүмкіндік береді. Осыған байланысты есептеу тапсырмаларын серверге жіберу және нәтижелерді пайдаланушыға қайтару жүйесін құру барысында ASP.NET MVC технологиясы қолданылды.

Аталған технологияларды пайдалану нәтижесінде жүйеге сәйкес модель, контроллер және көрсетілім, яғни Model, Controller және View файлдары құрылды. Бұл құрылым 33-суретте көрсетілген.



Сурет 33 – Серверде есептеулерді жүргізуге қашықтан қол жеткізу жүйесі архитектурасын құру процессі.

Жүйені пайдалану барысында барлық функционалдық мүмкіндіктер пайдаланушыға желілік хаттама арқылы қолжетімді болады. Бұл жүйенің негізгі артықшылығы — клиент тарапынан тек браузерді пайдалана отырып, есептеу тапсырмаларын сервердегі графикалық процессорда іске қосу мүмкіндігі. Пайдаланушы енгізген барлық деректер SQL Server басқаруымен деректер қорында сақталады.

SQL Server деректерді басқаруға, сақтауға және талдауға арналған маңызды корпоративтік қосымшалар үшін қолданылатын қуатты платформа болып табылады. Серверлік архитектураны ұйымдастыруда бұл платформа күрделі талдау жүргізуге, жоғары сенімділікті қамтамасыз етуге және деректерді өңдеудің күрделі құрылымы бар жүйелерді қолдауға мүмкіндік береді.

SQL Server жүйесінде SQLAgentUserRole, SQLAgentReaderRole және SQLAgentOperatorRole сияқты алдын ала анықталған деректер қоры рөлдері қолданылады. Бұл рөлдер msdb дерекқорында сервердің sysadmin алдын ала анықталған рөліне кірмейтін пайдаланушылар үшін SQL Server Agent қызметіне қолжетімділікті басқаруға мүмкіндік береді. Аталған рөлдермен қатар ішкі жүйелер мен тіркеу жазбалары деректер қоры әкімшілеріне әрбір есептеу кезеңін орындау үшін қажетті рұқсаттардың минималды және жеткілікті деңгейде берілуін қамтамасыз етеді.

Құрастырылатын жүйеде SQL Server платформасын қолдану келесі талаптарды орындауға бағытталды:

- дерекқорға мерзімді деректерді жазу
- тіркеу жазбаларын құру және басқару
- жүйені орталықтандырылған түрде басқару
- деректерге қолжетімділікті жеңілдету
- пайдаланушы жобаларын сақтау
- деректердің тұтастығы мен қауіпсіздігін қамтамасыз ету.

Бұл деректер қорын басқару жүйесі үлкен көлемдегі деректерді өңдеу үшін

қосымша технологиялармен кеңейтілуі мүмкін. SQL Server жүйесінің маңызды ерекшеліктерінің бірі — параллельді есептеулерді қолдауы. Аталған деректер қорын басқару жүйесінде көптеген операциялар параллель орындалады және SQL Server параллельді есептеу аппараттық құралдарының мүмкіндіктерін тиімді пайдаланады.

Мысалы, кестелерді секцияларға бөлу, яғни partitioning, және онлайн режимдегі параллельді индекстеу деректерді жылдам жүктеуге мүмкіндік береді. Индекстер мен үлкен көлемдегі деректерді секцияларға бөлу секцияланған сұраныстарды тиімді өңдеуді және сұраныстардың параллель орындалуын қамтамасыз етеді. Бұл үлкен дерекқорлардың өнімділігін арттыруға ықпал етеді.

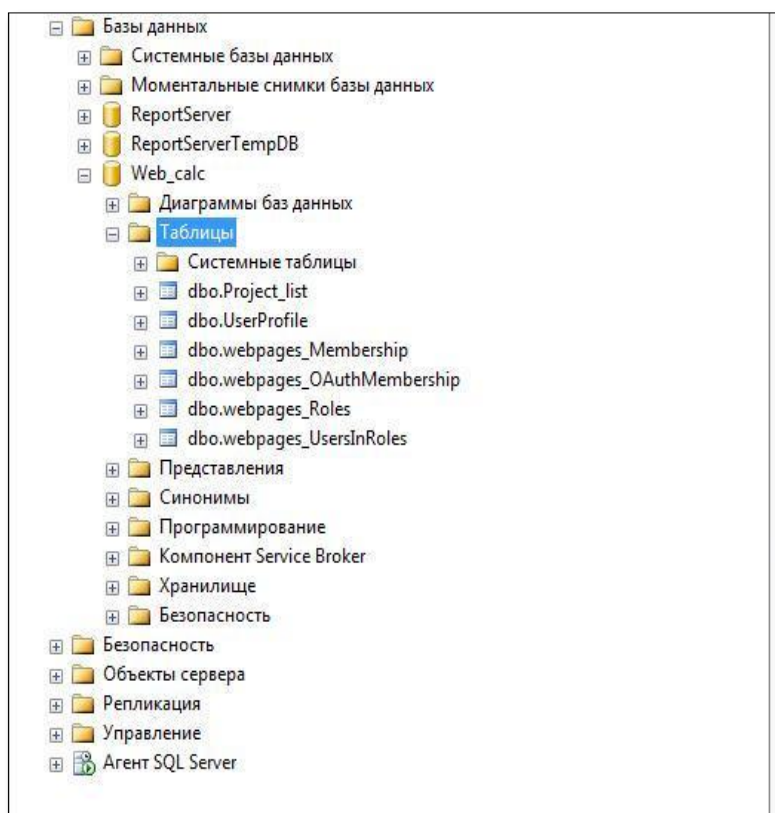
Сонымен қатар жеке секцияларды қалпына келтіру, қалпына келтіру уақытын қысқарту және деректерді физикалық файлдар топтарына бөлу үлкен деректер жиындарын басқаруды жеңілдетеді. Мұндай тәсіл «жылжымалы терезе» әдісі арқылы резервтік көшіру және деректерді сақтау үдерістерін тиімді ұйымдастыруға мүмкіндік береді.

Жүйені вертикалды масштабтау кезінде сервер бір уақытта көбірек параллель сұраныстарға қызмет көрсетуі қажет болады. Алайда маңызды операциялар ресурстардың жетіспеушілігінен немесе басқа процестердің әсерінен баяуламауы тиіс. Осы мақсатта SQL Server құрамында Resource Governor құралы қарастырылған.

Resource Governor құралы ресурстарды пайдалануға шектеу қоюға және әртүрлі тапсырмаларға басымдық беруге мүмкіндік береді. Оның көмегімен соңғы пайдаланушыға жүйенің тұрақты және болжамды жауап беруін қамтамасыз етуге болады. Бұл құрал пайдаланушы, қосымша, дерекқор және басқа да параметрлер негізінде дерекқор серверіне түсетін максималды жүктемені анықтайды. Сонымен қатар әрбір тапсырма үшін процессор уақыты мен жад көлемін пайдалануды шектей алады.

Осы мүмкіндік ресурстарды тиімді басқаруға және бір тапсырманың шамадан тыс жүктеме тудыруы арқылы басқа тапсырмалардың орындалуына кері әсер етуін болдырмауға жағдай жасайды. Ресурстарды алдын ала жоспарлау әртүрлі жүктеме жағдайларында жүйенің болжамды өнімділігін қамтамасыз етеді. Бұл деректер қызметіндегі ақауларды анықтау, жою және басқару үдерістерін жеңілдетеді.

SQL Server әртүрлі байланыс орнату технологиялары арқылы қосымшалар құруға мүмкіндік береді. Әсіресе .NET Framework платформасымен бірге қолданылған жағдайда қуатты және икемді қосымшаларды әзірлеуге болады. ADO.NET Entity Framework ортасында логикалық ақпараттық нысандармен жұмыс істеу мүмкіндігі қарастырылған. Сонымен қатар пайдаланушыларға деректерді өз құрылғыларына көшіріп, кейін оларды орталықтандырылған сервермен синхрондауға мүмкіндік беретін қосымшалар құруға болады. Бұл құрылым 34-суретте көрсетілген.

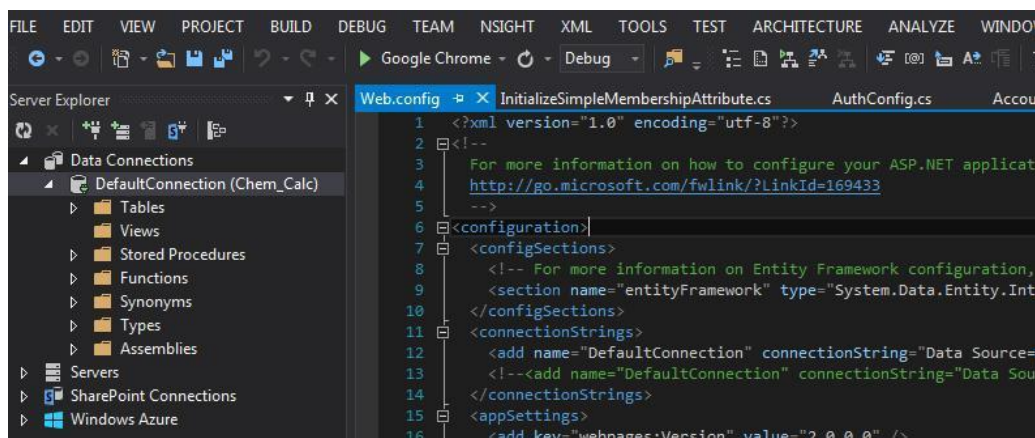


Сурет 34 - SQL SERVER дерекқорының құрылымы

Жану үдерісінде нақты отындарды модельдеуге арналған жүйені құру барысында Windows Server 2019 операциялық жүйесі негізінде SQL Server орнатылып, қажетті баптаулар жүргізілді. Жүйеде пайдаланушыларды тіркеу, жобаларды сақтау, реакциялар тізімін жүргізу және термодинамикалық параметрлерді дерекқорда сақтау мүмкіндіктері ұйымдастырылды. Бұл құрылым 23-суретте көрсетілген.

Деректерге қол жеткізу жылдамдығы мен жүйенің жалпы тиімділігін арттыру мақсатында дерекқордағы мәліметтер индекстелді және олардың сәйкес деректер типтері анықталды. Мұндай тәсіл сұраныстарды өңдеу уақытын қысқартуға және үлкен көлемдегі мәліметтермен жұмыс істеу кезінде жүйенің өнімділігін арттыруға мүмкіндік береді.

IIS веб-сервері, GPU-да орындалатын есептеу модулі және дерекқор жүйесі бір физикалық серверде орналасқандықтан, веб-қосымшада жергілікті сервермен байланыс орнатуға арналған Web.config файлындағы ConnectionString параметрі қолданылды. Бұл параметр қосымшаның SQL Server дерекқорына тікелей қосылуын қамтамасыз етеді және деректерді оқу, жазу, жаңарту сияқты операцияларды орындауға мүмкіндік береді. Аталған баптау 35-суретте көрсетілген.



Сурет 35 – Деректер қорын құру процесі

Бұл жағдайда бейнеадаптер, яғни GPU ядроларына пайдаланушылардың таратылған қолжетімділігін ұйымдастыру бірнеше пайдаланушының есептеу тапсырмаларын бір уақытта жоғары жылдамдықпен орындауға мүмкіндік береді.

CUDA технологиясы GPU үшін бағдарламалық код жазуға арналған C тілінің мүмкіндіктерін кеңейтеді. Бұл кеңейтулер nvcc.exe компиляторы арқылы өңделеді және графикалық процессор ресурстарын параллель есептеулерге тиімді пайдалануға мүмкіндік береді. CUDA технологиясы келесі негізгі мүмкіндіктерді қамтиды:

- функция спецификаторлары, яғни функциялардың қай құрылғыда және қандай тәртіппен орындалатынын анықтайтын белгілер;
- айнымалы спецификаторлары, олар қолданылатын GPU жадысының типін көрсетуге мүмкіндік береді;
- GPU ядросын, яғни kernel функциясын іске қосу спецификаторлары;
- ядро орындалу кезінде блоктар мен жіптерді, яғни blocks және threads құрылымдарын сәйкестендіруге арналған кіріктірілген айнымалылар;
- айнымалылардың қосымша типтері.

Аталған кеңейтулер бейнеадаптер ресурстарын параллель есептеу үдерістерінде тиімді пайдалануға жағдай жасайды. Бұл жұмыста CUDA мүмкіндіктері жану үдерісін модельдеу кезінде химиялық компоненттер концентрациясының уақыт бойынша өзгерісін есептеу үшін қолданылды.

Серверде деректерді өңдеу және есептеу үдерісі келесі негізгі кезеңдерден тұрады:

1. Есептеуге қажетті бастапқы деректерді алу;
2. Деректер көшірмесін GPU жадысына орналастыру;
3. Kernel функциясы арқылы GPU құрылғысында есептеулер жүргізу;
4. Есептелген нәтижелерді GPU жадысынан жедел жадыға, яғни RAM жадысына көшіру;
5. Нәтижелерді өңдеу және пайдаланушыға көрсету;
6. Қолданылған есептеу ресурстарын босату.

Бұл жұмыста бастапқы деректерді енгізу және алу үдерісі пайдаланушы тарапында жүзеге асырылады. Ал негізгі есептеу үдерісі IIS сервері арқылы

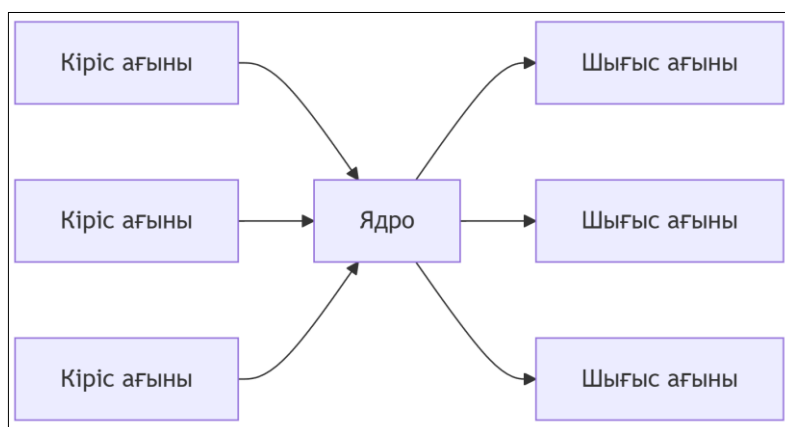
CUDA технологиясын пайдалану негізінде орындалады. Яғни пайдаланушы енгізген деректер серверге жіберіліп, одан кейін GPU жадысына көшіріледі және графикалық процессорда параллель түрде өңделеді.

Есептеу архитектурасын жобалау барысында CUDA бағдарламалау моделінің негізгі қағидалары ескерілді. Бағдарламалық код тізбекті және параллель бөліктерден тұрады. Тізбекті бөлігі CPU, яғни host құрылғысында орындалады. Ал массивті-параллель бөлігі GPU, яғни device құрылғысында іске асырылады.

Параллель есептеу бөлігі көптеген жіптер, яғни threads түрінде ұйымдастырылады. Жіптер өзара блоктарға, яғни blocks құрылымына біріктіріледі. Ал блоктар жиынтығы тор, яғни grid құрылымын құрайды. GPU ядросы, яғни kernel, осы блоктар торында орындалады. Мұндай құрылым үлкен көлемдегі біртекті есептеулерді бір уақытта орындауға мүмкіндік береді.

Серверде орындалатын есептеулер барысында деректердің бір бөлігі SQL Server дерекқорынан алынады. Ал жүйенің өнімділігін арттыру мақсатында кейбір деректер алдын ала дайындалған және жүктелген файлдардан оқылады. Бұдан кейін есептеу параметрлері, есептеу әдістері және бастапқы шарттар беріледі.

CUDA технологиясын қолдану кезінде есептеулер SIMD, яғни Single Instruction, Multiple Data принципіне негізделеді. Бұл принцип бойынша бір нұсқау бірнеше дерек элементіне бір уақытта қолданылады. Басқаша айтқанда, кіріс ретінде біртекті деректер ағыны беріледі, ал олардың әрқайсысы GPU жіптері арқылы тәуелсіз түрде өңделеді. Бұл тәсіл жану үдерісіндегі көптеген реакциялар мен компоненттер бойынша концентрация өзгерісін параллель есептеуге мүмкіндік береді. Аталған есептеу құрылымы 36-суретте көрсетілген.

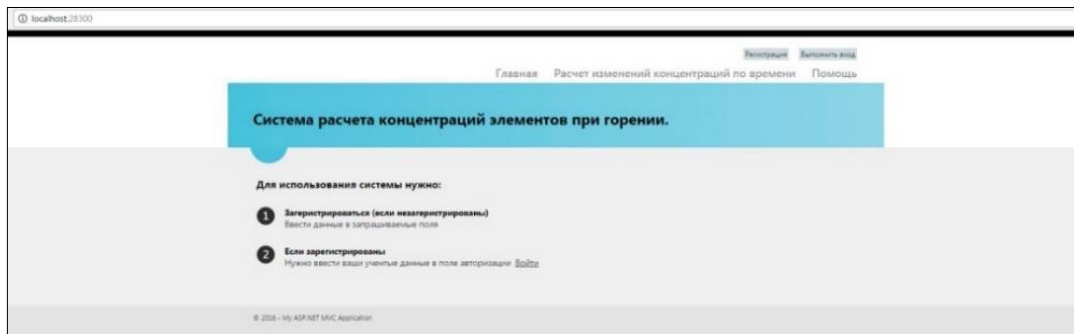


Сурет 36 – CUDA технологиясы жұмысы принципі

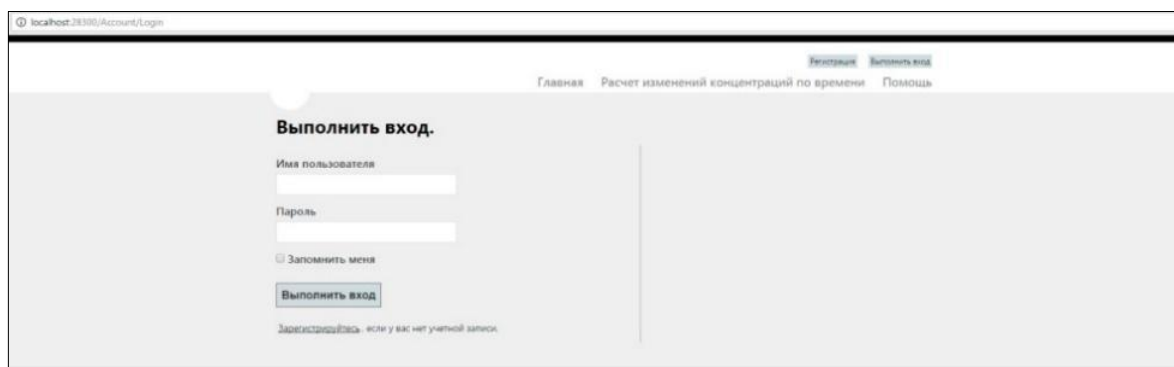
Есептеудің мұндай тәсілі бірнеше процесті параллель түрде орындауға мүмкіндік береді және бұл жалпы есептеу жылдамдығына кері әсерін тигізбейді. Параллель өңдеу нәтижесінде жүйе бір уақытта бірнеше есептеу тапсырмасын қабылдап, оларды тиімді түрде орындауға қабілетті болады.

Серверде есептеу амалдары толық орындалғаннан кейін бағдарлама алынған нәтижелерді дерекқорға сақтайды және пайдаланушыға жобаның нәтижелік көрінісін ұсынады. Бұл пайдаланушыға орындалған есептеулерді кейін қайта қарауға, салыстыруға және талдауға мүмкіндік береді.

Есептеу бағдарламасына қол жеткізу үшін клиент тарапынан веб-браузер ашылып, адрестік жолға жүйенің сәйкес мекенжайы енгізіледі. Осыдан кейін пайдаланушының басты интерфейсі жүктеледі. Аталған интерфейс 37-суретте көрсетілген.



Сурет 37 – Қолданушының басты интерфейсі



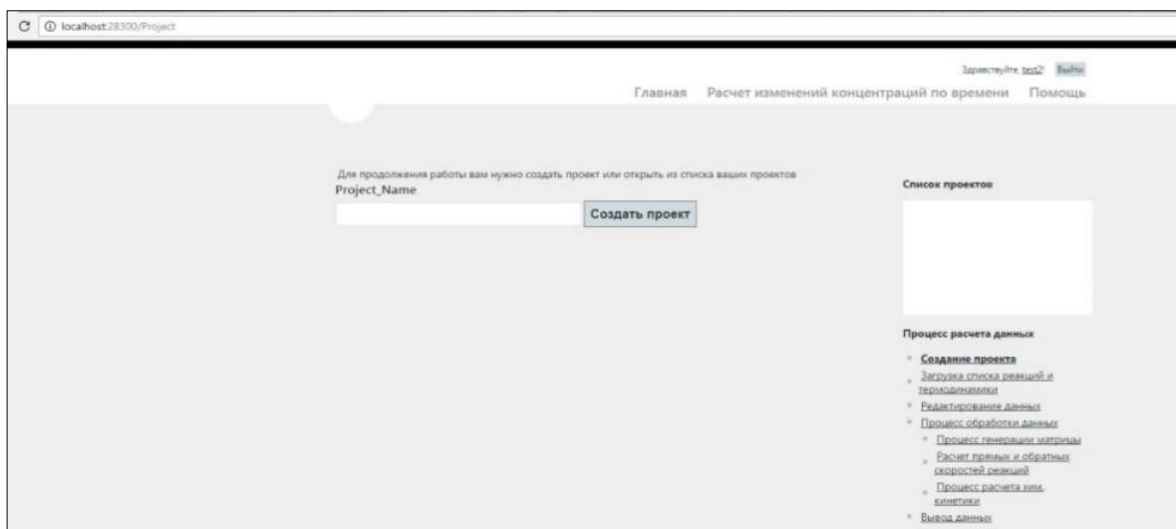
Сурет 38 - Қолданушының жүйеге кіру интерфейсі

Қолданушы есептеу амалдарын орындау үшін алдымен жүйеде тіркелуі қажет. Егер қолданушы бұрын тіркелген болса, онда ол авторизациядан өту арқылы жүйеге кіре алады. Пайдаланушы есептеу сұраныстарын және қажетті файлдарды серверге жіберетіндіктен, жүйені жобалау барысында қауіпсіздікті қамтамасыз ету мақсатында тикеттерді тексеру арқылы авторизациялау механизмі енгізілді. Бұл жүйеге кіру үдерісі серверлік архитектураны ұйымдастырудағы маңызды қауіпсіздік факторларының бірі болып табылады. Авторизациялау интерфейсі 38-суретте көрсетілген.

Қолданушының жүйедегі келесі әрекеттері ақпаратты енгізу және өңдеу кезеңдері бойынша жіктеледі:

- жоба құру
- термодинамикалық деректер мен реакциялар тізімін енгізу
- деректерді өңдеу және редакциялау
- деректерді бастапқы өңдеу үдерісі
- матрицаны генерациялау үдерісі
- реакциялардың тура және кері жылдамдықтарын есептеу

- химиялық кинетиканы есептеу үдерісі
- есептеу нәтижелерін шығару.

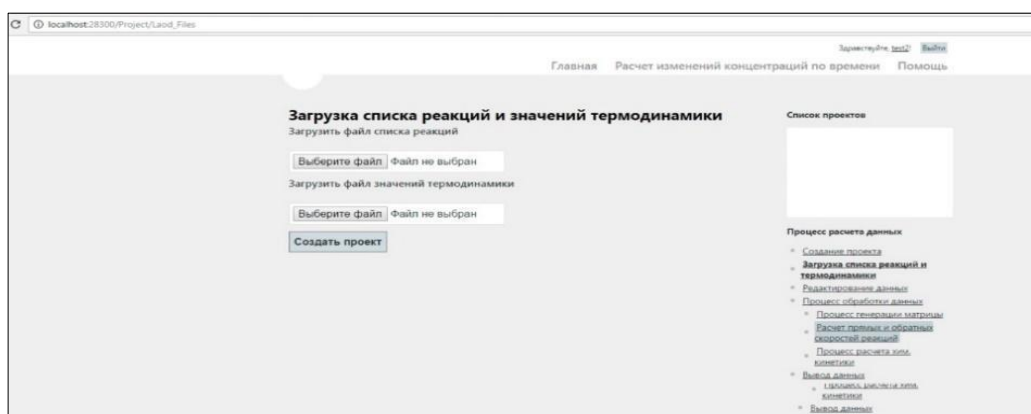


Сурет 39 - Жоба құру интерфейсі

Әрбір әрекет орындалған кезеңде жүйеде сұраныстың сәтті орындалуы немесе мәндерді өзгерту шарттары туралы ақпарат әрекеттер тарихында көрсетіліп отырады. Бұл пайдаланушыға орындалған операцияларды бақылауға және енгізілген өзгерістерді қадағалауға мүмкіндік береді.

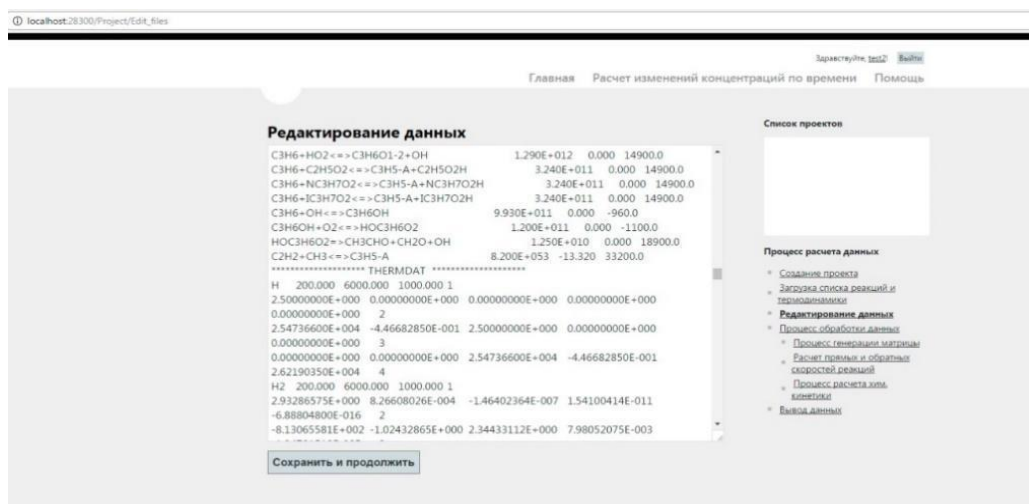
Жұмысты жалғастыру үшін қолданушы жаңа жоба құра алады немесе тізімде бұрын сақталған жобалары болса, солардың бірін аша алады. Жоба құру интерфейсі 39-суретте көрсетілген.

Жоба құрылғаннан кейін қолданушыға қажетті термодинамикалық деректерді және реакциялар тізімі бар файлдарды жүйеге енгізу мүмкіндігі беріледі. Бұл кезеңде пайдаланушы модельдеуге қажетті бастапқы мәліметтерді жүктеп, оларды әрі қарай есептеу үдерісінде қолдана алады. Аталған үдеріс 40-суретте көрсетілген.



Сурет 40 – Проект құру интерфейсі

Қолданушы енгізілген файлдарға қажет болса өзгерістер енгізе алады (сурет 41).

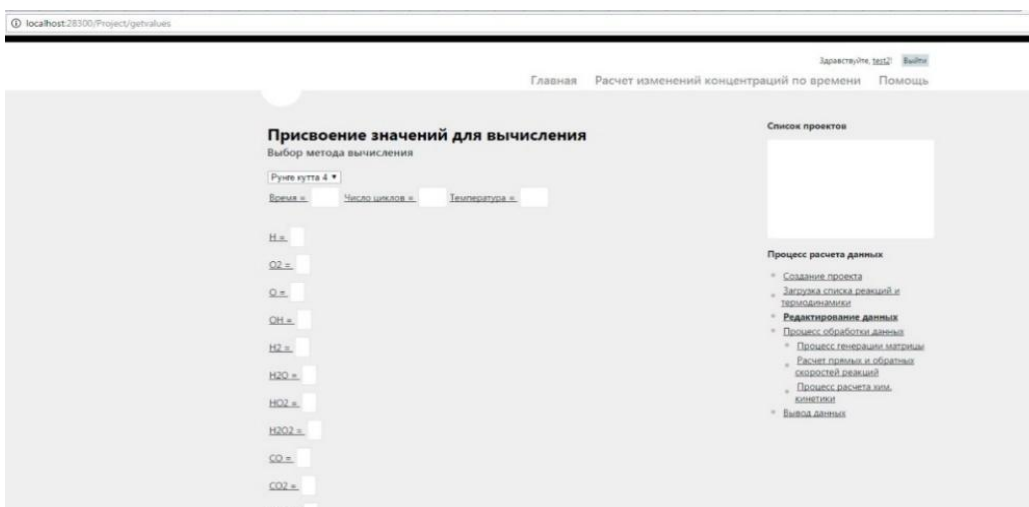


Сурет 41 – Енгізілген файлдар редакторы интерфейсі

Қолданушы енгізілген өзгерістерді сақтау және жұмысты жалғастыру үшін «Сақтау және жалғастыру» батырмасын басады. Осы әрекеттен кейін бағдарлама редактор аймағындағы термодинамикалық мәндер мен реакциялар тізіміне қатысты деректерді оқиды.

Келесі интерфейсте концентрация мәндерін енгізуге арналған реагенттер тізімі көрсетіледі. Бұл тізім бағдарлама тарапынан механизмді оқу барысында автоматты түрде генерацияланады. Яғни реакциялық механизм құрамындағы барлық реагенттер анықталып, олардың атаулары концентрация енгізу аймағына автоматты түрде шығарылады.

Сонымен қатар осы интерфейсте қолданушы бастапқы температураны, уақыт қадамын және есептеу циклдерінің санын енгізе алады. Бұл параметрлер концентрацияның уақыт бойынша өзгерісін есептеу үшін қолданылады. Бастапқы температура T , уақыт қадамы Δt , ал циклдер саны N арқылы белгіленеді. Аталған интерфейс 42-суретте көрсетілген.



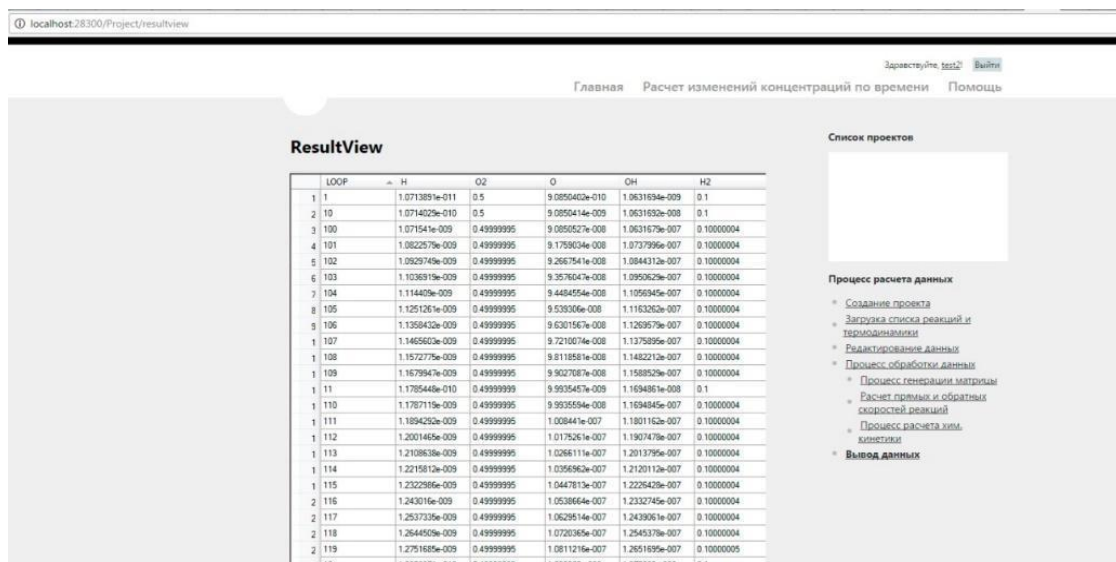
Сурет 42 – Деректерді енгізу және есептеулерді баптау интерфейсі.

Файлдар оқылғаннан кейін жүйе тура және кері реакциялардың

жылдамдықтарын есептейді, сондай-ақ есептеу нәтижелерін жазу және өңдеу үшін қажетті матрицаларды генерациялайды. Барлық бастапқы деректер жинақталып, алдын ала өңдеу кезеңдері аяқталғаннан кейін серверде графикалық процессорда есептеу жүргізуге арналған бағдарлама іске қосылады.

Осы кезеңде есептеу үшін қажетті барлық ақпарат GPU жадысына жіберіліп, есептеу үдерісі CUDA технологиясы негізінде орындалады. Нәтижесінде қолданушы есептеу амалдарын өзінің жергілікті компьютерінде емес, орталықтандырылған сервердің есептеу ресурстары арқылы орындайды. Бұл тәсіл пайдаланушы құрылғысына түсетін жүктемені азайтып, концентрацияның уақыт бойынша өзгеруін жоғары жылдамдықпен есептеуге мүмкіндік береді.

Есептеу аяқталғаннан кейін алынған нәтижелер серверде өңделіп, пайдаланушыға уақыт бойынша концентрация өзгерісінің деректері түрінде ұсынылады. Аталған үдеріс 43-суретте көрсетілген.



Сурет 43 – Нәтижені шығару интерфейсі.

ҚОРЫТЫНДЫ

Бұл диссертациялық жұмыста жану процестерінің кинетикалық теңдеулерін жоғары өнімді GPU архитектурасында есептеуге арналған параллель алгоритмдерді құру, оларды бағдарламалық түрде іске асыру және есептеу тиімділігін бағалау мәселелері қарастырылды. Жану процесін сипаттайтын химиялық кинетикалық механизмдер күрделі дифференциалдық теңдеулер жүйесінен тұратындықтан, оларды дәстүрлі CPU негізіндегі әдістермен есептеу көп уақытты және едәуір есептеу ресурстарын талап етеді. Осыған байланысты жұмыста CUDA технологиясын қолдану арқылы есептеу үдерісін жеделдету, реакциялар жүйесін GPU ағындары арасында параллель үлестіру және GPU жадысын тиімді пайдалану тәсілдері зерттелді.

Диссертациялық жұмысты орындау барысында келесідей нәтижелерге қол жеткізілді:

- Жану процестерінің кинетикалық модельдері талданып, оларды сандық шешу әдістерінің тиімділігі мен қолдану ерекшеліктері зерттелді. Кинетикалық теңдеулер жүйесін шешу үшін бесінші дәрежелі Рунге–Кутта әдісін қолдану негізделді.

- GPU архитектурасының ерекшеліктерін ескере отырып, химиялық кинетика есептерін параллель орындауға бейімделген жоғары өнімді алгоритмдер әзірленді. Реакциялар бойынша есептеу үдерісін CUDA ағындары арасында үлестіру тәсілі ұсынылды.

- CUDA технологиясы негізінде кинетикалық солвердің бағдарламалық іске асырылуы орындалып, GPU жадысын тиімді пайдалану, shared memory қолдану, atomicAdd операциялары және block reduction тәсілдері арқылы есептеу үдерісі оңтайландырылды. Нәтижесінде алгоритмдердің дәлдігі мен өнімділігі арттырылды.

- GPU-да есептеулерді параллель орындауға арналған модульді қамтитын бағдарламалық кешеннің архитектурасы дайындалып, құрылды және толық тестілеуден өткізілді. Алғашқы есептеу тәжірибелерінде 1006 реакциядан тұратын механизм үшін GPU алгоритмі CPU негізіндегі әдіспен салыстырғанда 17,5 есе жылдамдату көрсетті. Кейін GPU жадысын shared memory арқылы оңтайландыру нәтижесінде 493 компоненттен және 2716 реакциядан тұратын механизм үшін есептеу жылдамдығы CPU әдісімен салыстырғанда 154,89 есеге дейін артты. Бағдарламалық кешен валидациядан өтіп, авторлық құқықпен қорғалды.

Сондай-ақ, осы диссертациялық жұмыстың негізгі нәтижелері келесі жарияланымдарда көрсетілген:

- Сарсембаев М.С., Урмашев Б.А. Использование технологии CUDA для ускорения вычислений в задачах химической кинетики // Cybernetics and Systems Analysis. – Springer New York, 2019. Бұл жұмыста химиялық кинетика есептерін CUDA технологиясы арқылы жеделдету тәсілі қарастырылып, CPU және GPU есептеу тәсілдерінің салыстырмалы тиімділігі талданды.

- Сарсембаев М.С., Урмашев Б.А. Разработка кинетического механизма горения метана с образованием оксидов азота и полиароматических молекул // Материалы Международной конференции «Инновационное развитие и востребованность науки в современном Казахстане». – Алматы, 2017. Бұл жұмыста метанның жану механизмін құру және азот оксидтері мен полиароматтық молекулалардың түзілуін сипаттау мәселелері қарастырылды.

- Сарсембаев М.С., Макашев Е.П., Урмашев Б.А., Исахов А.А. Влияние сверхзвукового потока температуры на диффузионное горение водорода // Материалы III Международной конференции «Промышленные технологии и инженерия». – Шымкент, 2016. Бұл жұмыста сутегінің диффузиялық жануына температуралық және ағындық факторлардың әсері зерттелді.

- Сарсембаев М.С., Урмашев Б.А. Создание кинетического механизма горения метана с образованием оксидов азота // Материалы II Международной научно-практической конференции «Modern Methodology of Science and Education». – Варшава, Польша, 2018. Бұл жұмыста метан жануы кезінде азот оксидтерінің түзілуін сипаттайтын кинетикалық механизм құру мәселесі баяндалды.

Сонымен қатар, диссертациялық зерттеу нәтижелері негізінде **«Информационно-вычислительная система для автоматизированной генерации химических моделей, описывающих горение реальных топлив»** атты авторлық құқық объектісі тіркелді. Қазақстан Республикасының авторлық құқықпен қорғалатын объектілерге құқықтардың мемлекеттік тізіліміне мәліметтерді енгізу туралы **2020 жылғы 17 қарашада берілген №13255 куәлік** алынған.

Диссертациялық жұмыс аталған ғылыми жобалармен тікелей байланысты және химиялық модельдерді автоматтандырылған түрде құру, бұлттық ақпараттық-есептеу жүйелерін ұйымдастыру, жоғары өнімді есептеу әдістерін қолдану және кинетикалық теңдеулерді жеделдетіп есептеу мәселелері төмендегі ғылыми жобалардың негізінде орындалды :

- «Нақты отынның жануын сипаттайтын химиялық модельдерді автоматты генерациялауға арналған бұлттық ақпараттық-есептеу жүйесінің теориялық негіздерін және прототипін әзірлеу», ҒЗЖ №0115PK00770, 2015–2017 жж.;

- «Фармакокинетика мен фармакодинамика үлгілерін автоматты әзірлеу және оңтайландыру үшін бұлттық ақпараттық-есептеу кешенін жасау және енгізу», IRN жобасы: AP051329, 2018–2020 жж.

Аталған жобалар аясында алынған ғылыми-әдістемелік және бағдарламалық нәтижелер диссертациялық жұмыстың зерттеу бағытын қалыптастыруға негіз болды. Атап айтқанда, бұлттық ақпараттық-есептеу жүйесін ұйымдастыру тәжірибесі, химиялық модельдерді автоматтандыру қағидалары және жоғары өнімді есептеу тәсілдері осы жұмыста ұсынылған GPU негізіндегі кинетикалық солверді әзірлеуге, CUDA технологиясы арқылы химиялық кинетика есептерін жеделдетуге және бағдарламалық кешеннің архитектурасын құруға тікелей пайдаланылды.

Жалпы, диссертациялық жұмыста ұсынылған алгоритмдер мен бағдарламалық шешімдер күрделі химиялық кинетикалық механизмдерді қысқа уақытта есептеуге мүмкіндік береді. Алынған нәтижелер GPU архитектурасын және CUDA технологиясын жану процестерін модельдеу есептерінде тиімді қолдануға болатынын көрсетті. Ұсынылған тәсілдер баламалы отындарды зерттеу, энергетикалық қондырғыларды оңтайландыру, зиянды шығарындыларды бағалау және жоғары өнімді ғылыми есептеулер салаларында қолданылуы мүмкін.

ПАЙДАЛАНҒАН ӘДБИЕТТЕР ТІЗІМІ

1. Goodwin D. G., Moffat H. K., Schoegl I., Speth R. L., Weber B. W. Cantera: An Object-oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes. – Version 3.0.0. – Zenodo, 2023. – DOI: 10.5281/zenodo.8137090.
2. Kim K., Díaz-Ibarra O. H., Najm H. N., Zádor J., Safta C. TChem: A Performance Portable Parallel Software Toolkit for Complex Kinetic Mechanisms // Computer Physics Communications. – 2023. – Vol. 285. – Article 108628. – DOI: 10.1016/j.cpc.2022.108628.
3. Henry de Frahan M. T., Rood J. S., Day M. S., Sitaraman H., Yellapantula S., Perry B. A., Grout R. W., Almgren A. S., Zhang W., Bell J. B., Chen J. H. PeleC: An Adaptive Mesh Refinement Solver for Compressible Reacting Flows // The International Journal of High Performance Computing Applications. – 2023. – Vol. 37, No. 2. – P. 115–131. – DOI: 10.1177/10943420221121151.
4. Bielawski R., Barwey S., Prakash S., Raman V. Highly-Scalable GPU-Accelerated Compressible Reacting Flow Solver for Modeling High-Speed Flows // Computers & Fluids. – 2023. – Vol. 265. – Article 105972. – DOI: 10.1016/j.compfluid.2023.105972.
5. Piscaglia F., Ghioldi F., Onorati A. GPU Acceleration of CFD Simulations in OpenFOAM // Aerospace. – 2023. – Vol. 10, No. 9. – Article 792. – DOI: 10.3390/aerospace10090792.
6. Mezei R. A. Introduction to the Development of Web Applications Using ASP.NET Core MVC. – Cham: Springer, 2023. – DOI: 10.1007/978-3-031-30626-6.
7. Johnson M. S., Pang H.-W., Payne A. M., Green W. H. ReactionMechanismSimulator.jl: A Modern Approach to Chemical Kinetic Mechanism Simulation and Analysis // International Journal of Chemical Kinetics. – 2024. – Vol. 56. – P. 732–747. – DOI: 10.1002/kin.21753.
8. Jaouen N., Nguyen H.-T., Domingo P., Vervisch L. ORCh: A Package to Reduce and Optimize Chemical Kinetics. Application to Tetrafluoromethane Oxidation // SoftwareX. – 2024. – Vol. 27. – Article 101819. – DOI: 10.1016/j.softx.2024.101819.
9. Rao S., Chen B., Xu X. Heterogeneous CPU-GPU Parallelization for Modeling Supersonic Reacting Flows with Detailed Chemical Kinetics // Computer Physics Communications. – 2024. – Article 109195. – DOI: 10.1016/j.cpc.2024.109195.
10. Owen L. D., Ge W., Rieth M., Arienti M., Esclapez L., Soriano B. S., Mueller M. E., Day M. S., Sankaran R., Chen J. H. PeleMP: The Multiphysics Solver for the Combustion Pele Adaptive Mesh Refinement Code Suite // Journal of Fluids Engineering. – 2024. – Vol. 146, No. 4. – Article 041103. – DOI: 10.1115/1.4064494.
11. Ghioldi F., Piscaglia F., Onorati A. A Hybrid CPU-GPU Paradigm to Accelerate Reactive CFD Simulations // International Journal for Numerical Methods in Fluids. – 2024. – Vol. 96, No. 8. – P. 1461–1486. – DOI: 10.1002/flid.5297.

12. Tian X., Liu S., Fang J., Zhang Q., Feng Z., Fang Q. Experimental and Kinetic Modeling Investigation on Ignition Characteristics of Hydrogen: Effects of Methane Co-firing and Dilution Gas // *Progress in Reaction Kinetics and Mechanism*. – 2024. – Vol. 49. – Article e003. – DOI: 10.48130/prkm-0024-0002.
13. Impagnatiello M., Malé Q., Noiray N. Numerical Study of Ignition and Combustion of Hydrogen-Enriched Methane in a Sequential Combustor // *Flow, Turbulence and Combustion*. – 2024. – Vol. 112. – P. 1249–1273. – DOI: 10.1007/s10494-024-00540-8.
14. Pierro M., Urso J. J., Rahman R. K., Dennis C. W., Albright M. A., McGaunn J., Kinney C., Vasu S. S. Ignition Delay Times and Chemical Kinetic Model Validation for Hydrogen and Ammonia Blending With Natural Gas at Gas Turbine Relevant Conditions // *Journal of Engineering for Gas Turbines and Power*. – 2024. – Vol. 146, No. 6. – Article 061010. – DOI: 10.1115/1.4063789.
15. Balos C. J., Day M., Esclapez L., Felden A. M., Gardner D. J., Hassanaly M., Reynolds D. R., Rood J. S., Sexton J. M., Wimer N. T., Woodward C. S. SUNDIALS Time Integrators for Exascale Applications with Many Independent Systems of Ordinary Differential Equations // *The International Journal of High Performance Computing Applications*. – 2025. – Vol. 39, No. 1. – P. 123–146. – DOI: 10.1177/10943420241280060.
16. Danciu B. A., Frouzakis C. E. KinetiX: A Performance Portable Code Generator for Chemical Kinetics and Transport Properties // *Computer Physics Communications*. – 2025. – Vol. 310. – Article 109504. – DOI: 10.1016/j.cpc.2025.109504.
17. Pan K., Zhong S., Huang X., et al. A Reduced Chemical Kinetic Mechanism for CFD Modeling of Combustion Fueled by Ammonia and Its Blends with Methane/Hydrogen // *Chemical Engineering Journal*. – 2025. – Vol. 505. – Article 159253. – DOI: 10.1016/j.cej.2025.159253.
18. Koenig B. C., Kim S., Deng S., Green W. H. ChemKANs for Combustion Chemistry Modeling and Acceleration // *Physical Chemistry Chemical Physics*. – 2025. – Vol. 27, No. 33. – DOI: 10.1039/D5CP02009C.
19. Moure Á., Surapaneni A., Mira D. Optimized Workload Distribution for GPU-Accelerated Combustion Simulations in Heterogeneous CPU-GPU Architectures // *Computers & Fluids*. – 2025. – Vol. 302. – Article 106846. – DOI: 10.1016/j.compfluid.2025.106846.
20. Cai Y., Ban Y., Dang G., Gong T., Wang H., Zhang F., Chen Z. X. A GPU-Accelerated Stiff ODE Solver for Combustion CFD with Fully Analytical Chemical Kinetics and Heterogeneous CPU/GPU Integration // *Fuel*. – 2026. – Vol. 408. – Article 137627. – DOI: 10.1016/j.fuel.2025.137627.
21. Díaz-Ibarra O. H., et al. TChem-atm v2.0.0: Scalable Performance-Portable Simulation of Multiphase Atmospheric Chemistry on Heterogeneous Computing Architectures // *Geoscientific Model Development*. – 2026. – Vol. 19. – P. 1281–1306. – DOI: 10.5194/gmd-19-1281-2026.

22. Taboré A. KinOpt: A Python Package for Chemical Kinetics Analysis and Optimization // *Journal of Open Source Software*. – 2026. – Vol. 11, No. 118. – Article 7751. – DOI: 10.21105/joss.07751.
23. Zichi L., Barter D., Sivonxay E., Spotte-Smith E. W. C., Mohanakrishnan R. S., Chan E. M. RNMC: Kinetic Monte Carlo Implementations for Complex Reaction Networks // *Journal of Open Source Software*. – 2024. – Vol. 9, No. 104. – Article 7244. – DOI: 10.21105/joss.07244.
24. Cota F., et al. Development and Optimization of Chemical Kinetic Mechanisms for Combustion Modeling // *Energies*. – 2025. – Vol. 18, No. 16. – Article 4444. – DOI: 10.3390/en18164444.
25. Oleś S., et al. Assessment of Global and Detailed Chemical Kinetics in Hydrogen Combustion with Carbon Dioxide and Argon Diluents // *Energies*. – 2025. – Vol. 18, No. 13. – Article 3380. – DOI: 10.3390/en18133380.
26. Варнатц Ю., Маас У., Диббл Р. Горение физические и химические аспекты, моделирование, эксперименты, образование загрязняющих веществ. / Пер. с англ. Г.Л. Агафонова. Под ред. П.А. Власова. — М.: ФИЗМАТЛИТ, 2003. – 352 с.
27. Yang J., Xiong X., Yuan J., Zhang J., Cai L. Comparative Evaluation and Kinetic Analysis of Chemical Models for the Combustion of Oxymethylene Ethers // *Progress in Reaction Kinetics and Mechanism*. – 2026. – Vol. 51. – Article e006. – DOI: 10.48130/prkm-0025-0031.
28. Kathrotia T., et al. Combustion Kinetics of Alternative Fuels, Part IV: Extending the Model of Oxymethylene Ethers // *Combustion and Flame*. – 2025. – Vol. 271. – Article 113841. – DOI: 10.1016/j.combustflame.2024.113841.
29. Kumar R. S., et al. Methane Combustion Kinetics over Palladium-Based Catalysts // *Catalysts*. – 2024. – Vol. 14, No. 5. – Article 319. – DOI: 10.3390/catal14050319.
30. Michelbach C. A., et al. Automatic Mechanism Generation for the Combustion of Diethyl Ether // *International Journal of Chemical Kinetics*. – 2024. – DOI: 10.1002/kin.21705.
31. Mao R., Zhang M., Wang Y., Li H., Xu J., Dong X., Zhang Y., Chen Z. X. An Integrated Framework for Accelerating Reactive Flow Simulation Using GPU and Machine Learning Models // *Proceedings of the Combustion Institute*. – 2024. – Article 105512. – DOI: 10.1016/j.proci.2024.105512.
32. Holubinka V., Khudiyi A. Enhancing Database Query Performance: Analysis of Indexing Techniques // *Visnik Nàcional'nogo unìversitetu L'vìvs'ka politehnika. Seriâ Ìnformacìjnì sistemi ta merežì*. – 2024. – Vol. 15. – P. 65–73. – DOI: 10.23939/sisn2024.15.065.
33. Varga M. Web Programming and Multi-Tier Architecture of Web Applications // *TEM Journal*. – 2024. – Vol. 13, No. 4. – P. 3286–3294. – DOI: 10.18421/TEM134-63.
34. Necula S. Exploring the Model-View-Controller MVC Architecture: A Broad Analysis of Market and Technological Applications // *Preprints*. – 2024. – DOI: 10.20944/preprints202404.1860.v1.

35. Abbasi M., Bernardo M. V., Váz P., Silva J., Martins P. Optimizing Database Performance in Complex Event Processing through Indexing Strategies // *Data*. – 2024. – Vol. 9, No. 8. – Article 93. – DOI: 10.3390/data9080093.
36. Mushica F., Memeti A. A Comprehensive Analysis of Architectural Patterns in ASP.NET Core Web Application Development // *Journal of Natural Sciences and Mathematics of UT*. – 2024. – Vol. 9, No. 17–18. – P. 207–218. – DOI: 10.62792/ut.jnsm.v9.i17-18.p2816.
37. Malyshkin V. E., et al. *Parallel Computing Technologies. PaCT 2023*. – Cham: Springer, 2023. – DOI: 10.1007/978-3-031-41673-6.
38. Bienz A., et al. *High Performance Computing. ISC High Performance 2023*. – Cham: Springer, 2023. – DOI: 10.1007/978-3-031-32041-5.
39. Bienz A., et al. *High Performance Computing. ISC High Performance 2023 International Workshops*. – Cham: Springer, 2023. – DOI: 10.1007/978-3-031-40843-4.
40. Cano J., Danelutto M., et al. *Euro-Par 2023: Parallel Processing*. – Cham: Springer, 2023. – DOI: 10.1007/978-3-031-39698-4.
41. Tari Z., et al. *Algorithms and Architectures for Parallel Processing. ICA3PP 2023*. – Singapore: Springer, 2024. – DOI: 10.1007/978-981-97-0862-8.
42. Shi Y., Fu H., Tian Y., Krzhizhanovskaya V. V., Lees M. H., Dongarra J., Sloot P. M. A. *Computational Science – ICCS 2024*. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-63772-8.
43. Boulrier F., Mou C., Sadykov T. M., Vorozhtsov E. V. *Computer Algebra in Scientific Computing. CASC 2024*. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-69070-9.
44. Bienz A., et al. *High Performance Computing. ISC High Performance 2024 International Workshops*. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-73716-9.
45. Zhu T., Li J., Castiglione A. *Algorithms and Architectures for Parallel Processing. ICA3PP 2024*. – Singapore: Springer, 2025. – DOI: 10.1007/978-981-96-1548-3.
46. Caino-Lores S., Zeinalipour D., Doudali T. D., Singh D. E., Garzón G. E. M., Sousa L., Andrade D., Cucinotta T., D'Ambrosio D., Diehl P., Dolz M. F., Jukan A., Montella R., Nardelli M., Garcia-Gasulla M., Neuwirth S. *Euro-Par 2024: Parallel Processing Workshops*. – Cham: Springer, 2025. – DOI: 10.1007/978-3-031-90200-0.
47. Caino-Lores S., Zeinalipour D., Doudali T. D., Singh D. E., Garzón G. E. M., Sousa L., Andrade D., Cucinotta T., D'Ambrosio D., Diehl P., Dolz M. F., Jukan A., Montella R., Nardelli M., Garcia-Gasulla M., Neuwirth S. *Euro-Par 2024: Parallel Processing Workshops*. – Cham: Springer, 2025. – DOI: 10.1007/978-3-031-90203-1.
48. Neuwirth S., Paul A. K., Weinzierl T., Carson E. C. *High Performance Computing. ISC High Performance 2025 International Workshops*. – Cham: Springer, 2026. – DOI: 10.1007/978-3-032-07612-0.
49. Zhu T., Li J., Castiglione A. *Algorithms and Architectures for Parallel Processing. ICA3PP 2024. Proceedings, Part VI*. – Singapore: Springer, 2025. – DOI: 10.1007/978-981-96-1551-3.

50. Liu H., Ibrahim S., Rauber T. Algorithms and Architectures for Parallel Processing. ICA3PP 2025. Proceedings, Part I. – Singapore: Springer, 2026. – DOI: 10.1007/978-981-95-8399-7.
51. Liu H., Ibrahim S., Rauber T. Algorithms and Architectures for Parallel Processing. ICA3PP 2025. Proceedings, Part V. – Singapore: Springer, 2026. – DOI: 10.1007/978-981-95-8414-7.
52. Carretero J., Shende S., Garcia-Blas J., Brandic I., Olcoz K., Schreiber M. Euro-Par 2024: Parallel Processing. Proceedings, Part I. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-69577-3.
53. Carretero J., Shende S., Garcia-Blas J., Brandic I., Olcoz K., Schreiber M. Euro-Par 2024: Parallel Processing. Proceedings, Part III. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-69583-4.
54. Nagel W. E., Goehringer D., Diniz P. C. Euro-Par 2025: Parallel Processing. Proceedings, Part III. – Cham: Springer, 2025. – DOI: 10.1007/978-3-031-99872-0.
55. Niethammer C., Mix H., Nagel W. E., Resch M. M. Tools for High Performance Computing 2023. – Cham: Springer, 2026. – DOI: 10.1007/978-3-032-16397-4.
56. Nagel W. E., Kröner D. B., Resch M. M. High Performance Computing in Science and Engineering '23. – Cham: Springer, 2025. – DOI: 10.1007/978-3-031-91312-9.
57. Ranilla-Cortina S., Alonso-Jordá P., Vigo-Aguiar J., et al. The Evolution of High-Performance Computing: How AI and Quantum Computing Are Reshaping Supercomputing // The Journal of Supercomputing. – 2025. – Vol. 81. – Article 746. – DOI: 10.1007/s11227-025-07241-7.
58. Zou A., Li J., Kumar K., Anderson J. H. RTGPU: Real-Time GPU Scheduling of Hard Deadline Parallel Tasks // IEEE Transactions on Parallel and Distributed Systems. – 2023. – DOI: 10.1109/TPDS.2023.3235439.
59. Gheibi-Fetrat A., Ahmadi-Tonekaboni A., Koohi-Ronaghi F., Hajipour P., Babayan-Vanestan S., Fotouhi F., Mortazavian-Farsani E., Khajehpour-Dezfouli P., Safari S., Hessabi S., Sarbazi-Azad H. RTGPU: Real-Time Computing with Graphics Processing Units. – Cham: Springer, 2026. – DOI: 10.1007/978-3-032-11422-8.
60. Ferguson D., Pahl C., Helfert M. Cloud Computing and Services Science. CLOSER 2024. – Cham: Springer, 2026. – DOI: 10.1007/978-3-032-17286-0.
61. van Steen M., et al. Cloud Computing and Services Science. CLOSER 2023. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-68165-3.
62. Wang Y., et al. CLOUD Computing – CLOUD 2024. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-77153-8.
63. Zbakh M., Essaaidi M., Tadonki C., Touhafi A., Panda D. K. Artificial Intelligence and High Performance Computing in the Cloud: Research and Application Challenges. – Cham: Springer, 2025. – DOI: 10.1007/978-3-031-78698-3.
64. Lee R. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. – Cham: Springer, 2024. – DOI: 10.1007/978-3-031-56388-1.

65. Cao S., Zhang H., Liu H., Lyu Z., Li X., Zhang B., Han Y. Optimization of Kinetic Mechanism for Hydrogen Combustion Based on Machine Learning // *Frontiers of Chemical Science and Engineering*. – 2024. – Vol. 18, No. 11. – Article 1. – DOI: 10.1007/s11705-024-2487-0.
66. Saccone G., Marini M. Chemical Kinetic Analysis of High-Pressure Hydrogen Ignition and Combustion toward Green Aviation // *Aerospace*. – 2024. – Vol. 11, No. 2. – Article 112. – DOI: 10.3390/aerospace11020112.
67. Abbasi M., et al. Revisiting Database Indexing for Parallel and Accelerated Systems // *Information*. – 2024. – Vol. 15, No. 8. – Article 429. – DOI: 10.3390/info15080429.
68. Xu C. An Iterative Dynamic Chemical Stiffness Removal Method for Efficient Time Integration of Stiff Chemistry // *Flow*. – 2025. – DOI: 10.1007/s44270-024-00006-2.
69. Wang Y., Zeng Y., Deiterding R., Liang J. An Efficient GPU-Accelerated Adaptive Mesh Refinement Framework for High-Fidelity Compressible Reactive Flows Modeling // *Computer Physics Communications*. – 2026. – Vol. 318. – Article 109870. – DOI: 10.1016/j.cpc.2025.109870.
70. Croquevielle L. A., et al. Querying in Constant Expected Time with Learned Indexes // *28th International Conference on Database Theory, ICDT 2025*. – 2025. – Article 19. – DOI: 10.4230/LIPIcs.ICDT.2025.19.
71. Chakkappen S., et al. Automatic Indexing in Oracle // *Proceedings of the VLDB Endowment*. – 2025. – DOI: 10.14778/3750601.3750616.
72. Сарсембаев М.С., Урмашев Б.А. Использование технологии CUDA для ускорения вычислений в задачах химической кинетики // *Cybernetics and Systems Analysis*. – США: Springer New York, 2019.
73. Сарсембаев М. С., Урмашев Б. А., Темирбеков А. Н. Using the CUDA Technology to Accelerate Calculations in Problems of Chemical Kinetics // *Вестник Восточно-Казахстанского технического университета им. Д. Серикбаева*. – Казахстан, 2018. – № 3, т. 1, ч. 3. – С. 58–69.
74. Сарсембаев М. С., Турдалыулы М., Омарова П. Cloud Data-Processing System for the Automated Generation of Combustion Models // *International Journal of Mathematics and Physics*. – Казахстан, 2021. – № 7, т. 1. – С. 65–68.
75. Сарсембаев М. С., Урмашев Б. А., Сарсембаева Т. С., Мамырбаев О. Ж., Турдалыулы М. Using the CUDA Technology to Speed Up Computations in Problems of Chemical Kinetics // *Известия Национальной академии наук Республики Казахстан*. – Казахстан, 2021. – Т. 2. – С. 39–46.

ҚОСЫМША А.

Қосымшаның листингі Global memory көмегімен есептеу

//файл cudaRunRK4.cu

```
#include "cuda_runtime.h"
```

```
#include "cuda.h"
```

```
#include "device_launch_parameters.h"
```

```
#include "device_functions.h"
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
__device__ double atomicAdd_double(double *address, double val)
```

```
{
```

```
    unsigned long long int * address_as_ull = (unsigned long long int*)address;
```

```
    unsigned long long int old = *address_as_ull, assumed;
```

```
    do
```

```
    {
```

```
        assumed = old;
```

```
        old = atomicCAS(address_as_ull, assumed,
```

```
            __double_as_longlong(val +
```

```
            __longlong_as_double(assumed)));
```

```
    } while (assumed != old);
```

```
    return __longlong_as_double(old);
```

```
}
```

```
__global__ void cudaRungeCutta_1(int N, int M, double dt, int *left, int *right, double *c, double
```

```
*ccc, double *k_f, double *k_r, double *k1,
```

```
double *k2, double *k3, double *k4, double *k5, int k)
```

```
{
```

```
    int i = blockIdx.x * blockDim.x + threadIdx.x;
```

```
    if (i < N){
```

```
        k1[k] = 0.0;
```

```
        double mult1, mult2;
```

```
        mult1 = 1.0; mult2 = 1.0;
```

```
        for (int j = 0; j < M; j++)
```

```
        {
```

```
            mult1 = mult1 * pow(c[j], left[i*M + j]);
```

```
            mult2 = mult2 * pow(c[j], right[i*M + j]);
```

```
        }
```

```
        __syncthreads();
```

```
        atomicAdd_double(&k1[k], (1.0 / 3.0) * dt * (left[i*M + k] - right[i*M + k]) * (k_r[i] * mult2 - k_f[i] * mult1));
```

```
    }
```

```
}
```

```
__global__ void cudaRungeCutta_2(int N, int M, double dt, int *left, int *right, double *c, double
```

```
*ccc, double *k_f, double *k_r, double *k1,
```

```
double *k2, double *k3, double *k4, double *k5, int k)
```

```

{
int i = blockIdx.x * blockDim.x + threadIdx.x;
if (i < N)
{
k2[k] = 0.0;
double local_dc;
double mult1, mult2;
mult1 = 1.0; mult2 = 1.0;
for (int j = 0; j < M; j++)
{
if (j == k)
{
local_dc = c[k] + (i + 1) * k1[k];
mult1 = mult1 * pow(local_dc, left[i*M + j]);
mult2 = mult2 * pow(local_dc, right[i*M + j]);
}
else
{
mult1 = mult1 * pow(c[j], left[i*M + j]);
mult2 = mult2 * pow(c[j], right[i*M + j]);
}
}
__syncthreads();
atomicAdd_double(&k2[k], (1.0 / 3.0) * dt * (left[i*M + k] - right[i*M + k]) * (k_r[i] * mult2 - k_f[i] *
mult1));
__syncthreads();
if (i == N - 1) {
c[k] = c[k] + N * k1[k];
}
}
}

__global__ void cudaRungeCutta_3(int N, int M, double dt, int *left, int *right, double *c, double
*ccc, double *k_f, double *k_r, double *k1,
double *k2, double *k3, double *k4, double *k5, int k)
{
int i = blockIdx.x * blockDim.x + threadIdx.x;
if (i < N)
{
k3[k] = 0.0;
double local_dc;
double mult1, mult2;
__syncthreads();
mult1 = 1.0; mult2 = 1.0;
for (int j = 0; j < M; j++)
{
if (j == k)
{
local_dc = c[k] + (i + 1) * 0.5 * (k1[k] + k2[k]);
mult1 = mult1 * pow(local_dc, left[i*M + j]);

```

```

mult2 = mult2 * pow(local_dc, right[i*M + j]);
}
else
{
mult1 = mult1 * pow(c[j], left[i*M + j]);
mult2 = mult2 * pow(c[j], right[i*M + j]);
}
}
__syncthreads();
atomicAdd_double(&k3[k], (1.0 / 3.0) * dt * (left[i*M + k] - right[i*M + k]) * (k_r[i] * mult2 - k_f[i] *
mult1));
if (i == N - 1) {
c[k] = c[k] + N * 0.5 * (k1[k] + k2[k]);
}
}
}

__global__ void cudaRungeCutta_4(int N, int M, double dt, int *left, int *right, double *c, double
*ccc, double *k_f, double *k_r, double *k1,
double *k2, double *k3, double *k4, double *k5, int k)
{
int i = blockIdx.x * blockDim.x + threadIdx.x;
if (i < N)
{
k4[k] = 0.0;
double local_dc;
double mult1, mult2;
mult1 = 1.0; mult2 = 1.0;
__syncthreads();
for (int j = 0; j < M; j++)
{
if (j == k)
{
local_dc = c[k] + (i + 1) * ((3.0 / 8.0) * k1[k] + (9.0 / 8.0) * k3[k]);
mult1 = mult1 * pow(local_dc, left[i*M + j]);
mult2 = mult2 * pow(local_dc, right[i*M + j]);
}
else
{
mult1 = mult1 * pow(c[j], left[i*M + j]);
mult2 = mult2 * pow(c[j], right[i*M + j]);
}
}
__syncthreads();
atomicAdd_double(&k4[k], (1.0 / 3.0) * dt * (left[i*M + k] - right[i*M + k]) * (k_r[i] * mult2 - k_f[i] *
mult1));
__syncthreads();
if (i == N - 1)
{
c[k] = c[k] + N * ((3.0 / 8.0) * k1[k] + (9.0 / 8.0) * k3[k]);
}
}
}

```

```

}
}
}
__global__ void cudaRungeCutta_5(int N, int M, double dt, int *left, int *right, double *c, double
*ccc, double *k_f, double *k_r, double *k1,
double *k2, double *k3, double *k4, double *k5, int k)
{
int i = blockIdx.x * blockDim.x + threadIdx.x;
if (i < N)
{
k5[k] = 0.0;
double local_dc;
double mult1, mult2;
mult1 = 1.0; mult2 = 1.0;
__syncthreads();
for (int j = 0; j < M; j++)
{
if (j == k)
{
local_dc = c[k] + (i + 1) * ((3.0 / 2.0)* k1[k] - (9.0 / 2.0) * k3[k] + 6.0 * k4[k]);
mult1 = mult1 * pow(local_dc, left[i*M + j]);
mult2 = mult2 * pow(local_dc, right[i*M + j]);
}
else
{
mult1 = mult1 * pow(c[j], left[i*M + j]);
mult2 = mult2 * pow(c[j], right[i*M + j]);
}
}
__syncthreads();
atomicAdd_double(&k5[k], (1.0 / 3.0) * dt * (left[i*M + k] - right[i*M + k]) * (k_r[i] * mult2 - k_f[i] *
mult1));
__syncthreads();
if (i == N - 1)
{
c[k] = c[k] + N * ((3.0 / 2.0)* k1[k] - (9.0 / 2.0) * k3[k] + 6.0 * k4[k]);
ccc[k] = c[k] + 0.5 * (k1[k] + 4.0 * k4[k] + k5[k]);
}
}
}
__global__ void cudaRungeCutta_new(double *c, double *ccc)
{
int i = blockIdx.x * blockDim.x + threadIdx.x;
c[i] = ccc[i];
}
int main(int argc, char * argv[])
{
int N = atoi(argv[1]);
int M = atoi(argv[2]);

```

```

int LOOP = atoi(argv[3]);
double dt = atof(argv[4]);
string fname = argv[5];
string fnameRes = "Output/resultGPURK5" + fname + ".dat";
string fnameInput = "Output/inputParamsGPURK5" + fname + ".dat";
int deviceCount = 0;
cudaGetDeviceCount(&deviceCount);
if (deviceCount < 1)
{
    return -1;
}
ofstream result;      result.open(fnameRes, ios::binary | ios::out);
ofstream input;       input.open(fnameInput, ios::binary | ios::out);
cudaDeviceProp deviceProp;
input << "***** GPU *****\nDevice count: " << deviceCount << endl;
input << "GPU properties:" << endl;
for (int i = 0; i < deviceCount; i++)
{
    cudaGetDeviceProperties(&deviceProp, i);
    input << " Device name: " << deviceProp.name << endl;
    input << " Total global memory: " << deviceProp.totalGlobalMem << endl;
    input << " Shared memory per block: " << deviceProp.sharedMemPerBlock << endl;
    input << " Registers per block: " << deviceProp.regsPerBlock << endl;
    input << " Warp size: " << deviceProp.warpSize << endl;
    input << " Memory pitch: " << deviceProp.memPitch << endl;
    input << " Max threads per block: " << deviceProp.maxThreadsPerBlock << endl;
    input << " Max threads dimensions:\n x = " << deviceProp.maxThreadsDim[0] << "\n y = " <<
deviceProp.maxThreadsDim[1] << "\n z = " << deviceProp.maxThreadsDim[2] << endl;
    input << " Max grid size:\n x = " << deviceProp.maxGridSize[0] << "\n y = " <<
deviceProp.maxGridSize[1] << "\n z = " << deviceProp.maxGridSize[2] << endl;
    input << " Clock rate: " << deviceProp.clockRate << endl;
    input << " Total constant memory: " << deviceProp.totalConstMem << endl;
    input << " Compute capability:" << deviceProp.major<< "."<<deviceProp.minor << endl;
    input << " Texture alignment:" << deviceProp.textureAlignment << endl;
    input << " Device overlap: " << deviceProp.deviceOverlap << endl;
    input << " Multiprocessor count: " << deviceProp.multiProcessorCount << endl;
    input << " Kernel execution timeout enabled: ";
    if (deviceProp.kernelExecTimeoutEnabled)
        input << "true" << endl;
    else
        input << "false" << endl;
}
input << "*****\n";
ifstream k_f("Input/k_f" + fname + ".dat", ios::binary | ios::in);
ifstream k_r("Input/k_r" + fname + ".dat", ios::binary | ios::in);
ifstream conc("Input/conc" + fname + ".dat", ios::binary | ios::in);
ifstream left("Input/left" + fname + ".dat", ios::binary | ios::in);
ifstream right("Input/right" + fname + ".dat", ios::binary | ios::in);
double *hc2 = new double[M];

```

```

double *hk_f2 = new double[N];
double *hk_r2 = new double[N];
int *hleft2 = new int[N * M];
int *hright2 = new int[N * M];
double *cnew = new double[M];
double darg = 0.0;
int iarg = 0;
int k = 0;
k = 0; if (!k_f) { return -1; }      while (k_f >> darg) { hk_f2[k] = darg;      k++; } k_f.close();
k = 0; if (!k_r){ return -1; }      while (k_r >> darg) { hk_r2[k] = darg;
      k++; } k_r.close();
k = 0; if (!conc){ return -1; }      while (conc >> darg) { hc2[k] = darg;      k++; }
conc.close();
k = 0; if (!left){ return -1; }      while (left >> iarg) { hleft2[k] = iarg; k++; } left.close();
k = 0; if (!right){ return -1; }     while (right >> iarg){ hright2[k] = iarg;      k++; }
right.close();
input << "Method: 'Runge-Kutta 5'" << endl;
input << "LOOP = " << LOOP << endl;
input << "Reactions: " << N << endl << "Reagents: " << M << endl;
input << "Time step = " << dt << endl;
int *dleft, *dright;
double *dc, *dccc, *dk_f, *dk_r;
double *k1, *k2, *k3, *k4, *k5;
cudaMalloc((void**)&dk_f, N * sizeof(double));
cudaMalloc((void**)&dk_r, N * sizeof(double));
cudaMalloc((void**)&dleft, sizeof(int)* N * M);
cudaMalloc((void**)&dright, sizeof(int)* N * M);
cudaMalloc((void**)&dc, M * sizeof(double));
cudaMalloc((void**)&dccc, M * sizeof(double));
cudaMalloc((void**)&k1, M*sizeof(double));
cudaMalloc((void**)&k2, M*sizeof(double));
cudaMalloc((void**)&k3, M*sizeof(double));
cudaMalloc((void**)&k4, M*sizeof(double));
cudaMalloc((void**)&k5, M*sizeof(double));
cudaMemcpy(dk_f, hk_f2, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(dk_r, hk_r2, N * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(dleft, hleft2, N * M * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dright, hright2, N * M * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dc, hc2, M * sizeof(double), cudaMemcpyHostToDevice);
float gpuTime = 0.0;
cudaEvent_t start_dev, stop_dev; cudaEventCreate(&start_dev);
      cudaEventCreate(&stop_dev);
int gsize = N / 1024 + 1;
dim3 BLOCK_SIZE(1024);
dim3 GRID_SIZE(gsize);
dim3 BLOCK_SIZE1(M);
dim3 GRID_SIZE1(1);
k = 1;
cudaEventRecord(start_dev, 0);

```

```

do
{
for (int jj = 0; jj < M; jj++)
{
cudaRungeCutta_1 << <GRID_SIZE, BLOCK_SIZE >> >(N, M, dt, dleft, dright, dc, dccc, dk_f, dk_r, k1,
k2, k3, k4, k5, jj);
cudaRungeCutta_2 << <GRID_SIZE, BLOCK_SIZE >> >(N, M, dt, dleft, dright, dc, dccc, dk_f, dk_r, k1,
k2, k3, k4, k5, jj);
cudaRungeCutta_3 << <GRID_SIZE, BLOCK_SIZE >> >(N, M, dt, dleft, dright, dc, dccc, dk_f, dk_r, k1,
k2, k3, k4, k5, jj);
cudaRungeCutta_4 << <GRID_SIZE, BLOCK_SIZE >> >(N, M, dt, dleft, dright, dc, dccc, dk_f, dk_r, k1,
k2, k3, k4, k5, jj);
cudaRungeCutta_5 << <GRID_SIZE, BLOCK_SIZE >> >(N, M, dt, dleft, dright, dc, dccc, dk_f, dk_r, k1,
k2, k3, k4, k5, jj);
}
cudaRungeCutta_new << <GRID_SIZE1, BLOCK_SIZE1 >> >(dc, dccc);
cudaMemcpy(hc2, dccc, M * sizeof(double), cudaMemcpyDeviceToHost);
result << k << " ";    for (int i = 0; i < M; i++)    { result << setprecision(8) << hc2[i] << " "; }
result << endl;
k++;
} while (k <= LOOP);
cudaEventRecord(stop_dev, 0);
cudaEventSynchronize(stop_dev);
cudaEventElapsedTime(&gpuTime, start_dev, stop_dev);
result.close();
input << "GPU calc. time: " << setw(6) << gpuTime << " msec" << endl;
input.close();
cudaEventDestroy(start_dev);
cudaEventDestroy(stop_dev);
cudaFree(dleft);
cudaFree(dright);
cudaFree(dk_f);
cudaFree(dk_r);
cudaFree(dc);
cudaFree(k1);
cudaFree(k2);
cudaFree(k3);
cudaFree(k4);
cudaFree(k5);
cudaDeviceReset();
return 0;
}

```

ҚОСЫМША Б

Қосымшаның листингі shared memory көмегімен есептеу

```
// cudaRunRK5_shared_memory.cu
// CUDA-нұсқа: Runge–Kutta 5 + shared memory + block reduction
// Негізі: бастапқы cudaRunRK4.cu файлының құрылымы сақталды.
// Өзгеріс: әр потоктың global atomicAdd жасауын азайту үшін block reduction қолданылды.
```

```
#include "cuda_runtime.h"
#include "cuda.h"
#include "device_launch_parameters.h"
```

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
```

```
using namespace std;
```

```
// -----
// Double үшін atomicAdd.
// Compute capability >= 6.0 болса, CUDA native atomicAdd қолданады.
// Ескі GPU үшін atomicCAS арқылы іске асады.
// -----
__device__ double atomicAdd_double(double* address, double val)
{
    #if __CUDA_ARCH__ >= 600
        return atomicAdd(address, val);
    #else
        unsigned long long int* address_as_ull = (unsigned long long int*)address;
        unsigned long long int old = *address_as_ull, assumed;
        do
        {
            assumed = old;
            old = atomicCAS(address_as_ull, assumed,
                __double_as_longlong(val + __longlong_as_double(assumed)));
        } while (assumed != old);
        return __longlong_as_double(old);
    #endif
}

// -----
// Бір ғана k элементі үшін stage массивін нөлдеу.
// Мысалы: k1[k] = 0.0
// -----
__global__ void clear_stage_value(double* arr, int k)
{
    if (blockIdx.x == 0 && threadIdx.x == 0)
    {
        arr[k] = 0.0;
    }
}
```

```

    }
}

// -----
// c = ccc көшіру
// -----
__global__ void cudaRungeCutta_new_shared(double* c, const double* ccc, int M)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < M)
    {
        c[i] = ccc[i];
    }
}

// -----
// Stage нәтижесінен кейін c[k] немесе ccc[k] жаңарту.
// Бұл бөлек kernel ретінде жасалды, себебі блоктар арасында __syncthreads()
// синхронизация жасай алмайды.
// -----
__global__ void update_after_stage_2(double* c, const double* k1, int N, int k)
{
    if (blockIdx.x == 0 && threadIdx.x == 0)
    {
        c[k] = c[k] + N * k1[k];
    }
}

__global__ void update_after_stage_3(double* c, const double* k1, const double* k2, int N, int k)
{
    if (blockIdx.x == 0 && threadIdx.x == 0)
    {
        c[k] = c[k] + N * 0.5 * (k1[k] + k2[k]);
    }
}

__global__ void update_after_stage_4(double* c, const double* k1, const double* k3, int N, int k)
{
    if (blockIdx.x == 0 && threadIdx.x == 0)
    {
        c[k] = c[k] + N * ((3.0 / 8.0) * k1[k] + (9.0 / 8.0) * k3[k]);
    }
}

__global__ void update_after_stage_5(double* c, double* ccc,
                                     const double* k1, const double* k4, const double* k5,
                                     const double* k3, int N, int k)
{
    if (blockIdx.x == 0 && threadIdx.x == 0)
    {
        c[k] = c[k] + N * ((3.0 / 2.0) * k1[k] - (9.0 / 2.0) * k3[k] + 6.0 * k4[k]);
        ccc[k] = c[k] + 0.5 * (k1[k] + 4.0 * k4[k] + k5[k]);
    }
}

```

```

    }
}

// -----
// Shared memory қолданылатын жалпы RK stage kernel.
//
// stage = 1, 2, 3, 4, 5
//
// shared memory құрылымы:
// s_c[0 ... M-1]          -> концентрациялар
// s_reduce[0 ... blockDim.x-1] -> блок ішіндегі редукция
//
// Бұрын әр поток atomicAdd_double жасайтын.
// Енді әр блок алдымен shared memory ішінде қосынды жинайды,
// содан кейін әр блоктан тек 1 atomicAdd_double жасалады.
// -----
__global__ void cudaRungeCutta_stage_shared(
    int stage,
    int N,
    int M,
    double dt,
    const int* left,
    const int* right,
    const double* c,
    const double* k_f,
    const double* k_r,
    double* k1,
    double* k2,
    double* k3,
    double* k4,
    double* k5,
    int k)
{
    extern __shared__ double shared[];

    double* s_c = shared;
    double* s_reduce = &shared[M];

    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    // Концентрация массивін global memory-ден shared memory-ге көшіру.
    for (int j = tid; j < M; j += blockDim.x)
    {
        s_c[j] = c[j];
    }

    __syncthreads();

    double value = 0.0;

    if (i < N)

```

```

{
    double mult1 = 1.0;
    double mult2 = 1.0;

    for (int j = 0; j < M; j++)
    {
        double cj = s_c[j];

        if (j == k)
        {
            if (stage == 2)
            {
                cj = s_c[k] + (i + 1) * k1[k];
            }
            else if (stage == 3)
            {
                cj = s_c[k] + (i + 1) * 0.5 * (k1[k] + k2[k]);
            }
            else if (stage == 4)
            {
                cj = s_c[k] + (i + 1) * ((3.0 / 8.0) * k1[k] + (9.0 / 8.0) * k3[k]);
            }
            else if (stage == 5)
            {
                cj = s_c[k] + (i + 1) * ((3.0 / 2.0) * k1[k] - (9.0 / 2.0) * k3[k] + 6.0 * k4[k]);
            }
        }

        mult1 = mult1 * pow(cj, left[i * M + j]);
        mult2 = mult2 * pow(cj, right[i * M + j]);
    }

    value = (1.0 / 3.0) * dt *
        (left[i * M + k] - right[i * M + k]) *
        (k_r[i] * mult2 - k_f[i] * mult1);
}

// Block reduction in shared memory
s_reduce[tid] = value;
__syncthreads();

for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
{
    if (tid < stride)
    {
        s_reduce[tid] += s_reduce[tid + stride];
    }
    __syncthreads();
}

// Әр блоктан тек бір atomicAdd_double.
if (tid == 0)

```

```

{
    if (stage == 1)
    {
        atomicAdd_double(&k1[k], s_reduce[0]);
    }
    else if (stage == 2)
    {
        atomicAdd_double(&k2[k], s_reduce[0]);
    }
    else if (stage == 3)
    {
        atomicAdd_double(&k3[k], s_reduce[0]);
    }
    else if (stage == 4)
    {
        atomicAdd_double(&k4[k], s_reduce[0]);
    }
    else if (stage == 5)
    {
        atomicAdd_double(&k5[k], s_reduce[0]);
    }
}
}

// -----
// CUDA қателерін тексеру
// -----
static void checkCuda(cudaError_t err, const char* msg)
{
    if (err != cudaSuccess)
    {
        cerr << "CUDA error: " << msg << " : " << cudaGetErrorString(err) << endl;
        exit(EXIT_FAILURE);
    }
}

static void checkLastKernel(const char* name)
{
    cudaError_t err = cudaGetLastError();
    if (err != cudaSuccess)
    {
        cerr << "Kernel launch error in " << name << " : " << cudaGetErrorString(err) << endl;
        exit(EXIT_FAILURE);
    }
}

// -----
// main
// Аргументтер бастапқы кодтағыдай:
// argv[1] = N    реакция саны
// argv[2] = M    компонент/species саны
// argv[3] = LOOP уақыт бойынша қадам саны

```

```

// argv[4] = dt    уақыт қадамы
// argv[5] = fname  файл атының қосымшасы
// -----
int main(int argc, char* argv[])
{
    if (argc < 6)
    {
        cerr << "Usage: " << argv[0] << " N M LOOP dt fname" << endl;
        return -1;
    }

    int N = atoi(argv[1]);
    int M = atoi(argv[2]);
    int LOOP = atoi(argv[3]);
    double dt = atof(argv[4]);
    string fname = argv[5];

    string fnameRes = "Output/resultGPURK5_shared" + fname + ".dat";
    string fnameInput = "Output/inputParamsGPURK5_shared" + fname + ".dat";

    int deviceCount = 0;
    checkCuda(cudaGetDeviceCount(&deviceCount), "cudaGetDeviceCount");

    if (deviceCount < 1)
    {
        cerr << "CUDA device not found." << endl;
        return -1;
    }

    ofstream result;
    result.open(fnameRes.c_str(), ios::binary | ios::out);

    ofstream input;
    input.open(fnameInput.c_str(), ios::binary | ios::out);

    if (!result || !input)
    {
        cerr << "Cannot open output files." << endl;
        return -1;
    }

    cudaDeviceProp deviceProp;
    input << "***** GPU SHARED MEMORY VERSION *****\n";
    input << "Device count: " << deviceCount << endl;
    input << "GPU properties:" << endl;

    for (int dev = 0; dev < deviceCount; dev++)
    {
        checkCuda(cudaGetDeviceProperties(&deviceProp, dev), "cudaGetDeviceProperties");

        input << " Device name: " << deviceProp.name << endl;
        input << " Total global memory: " << deviceProp.totalGlobalMem << endl;
    }
}

```

```

input << " Shared memory per block: " << deviceProp.sharedMemPerBlock << endl;
input << " Registers per block: " << deviceProp.regsPerBlock << endl;
input << " Warp size: " << deviceProp.warpSize << endl;
input << " Memory pitch: " << deviceProp.memPitch << endl;
input << " Max threads per block: " << deviceProp.maxThreadsPerBlock << endl;
input << " Max threads dimensions:\n x = " << deviceProp.maxThreadsDim[0]
    << "\n y = " << deviceProp.maxThreadsDim[1]
    << "\n z = " << deviceProp.maxThreadsDim[2] << endl;
input << " Max grid size:\n x = " << deviceProp.maxGridSize[0]
    << "\n y = " << deviceProp.maxGridSize[1]
    << "\n z = " << deviceProp.maxGridSize[2] << endl;
input << " Clock rate: " << deviceProp.clockRate << endl;
input << " Total constant memory: " << deviceProp.totalConstMem << endl;
input << " Compute capability: " << deviceProp.major << "." << deviceProp.minor << endl;
input << " Multiprocessor count: " << deviceProp.multiProcessorCount << endl;
input << " Kernel execution timeout enabled: "
    << (deviceProp.kernelExecTimeoutEnabled ? "true" : "false") << endl;
}

input << "*****\n";

ifstream file_k_f(("Input/k_f" + fname + ".dat").c_str(), ios::binary | ios::in);
ifstream file_k_r(("Input/k_r" + fname + ".dat").c_str(), ios::binary | ios::in);
ifstream file_conc(("Input/conc" + fname + ".dat").c_str(), ios::binary | ios::in);
ifstream file_left(("Input/left" + fname + ".dat").c_str(), ios::binary | ios::in);
ifstream file_right(("Input/right" + fname + ".dat").c_str(), ios::binary | ios::in);

if (!file_k_f || !file_k_r || !file_conc || !file_left || !file_right)
{
    cerr << "Cannot open one or more input files." << endl;
    return -1;
}

double* hc2 = new double[M];
double* hk_f2 = new double[N];
double* hk_r2 = new double[N];
int* hleft2 = new int[N * M];
int* hright2 = new int[N * M];

double darg = 0.0;
int iarg = 0;
int k = 0;

k = 0;
while (file_k_f >> darg && k < N)
{
    hk_f2[k++] = darg;
}
file_k_f.close();

k = 0;
while (file_k_r >> darg && k < N)

```

```

{
    hk_r2[k++] = darg;
}
file_k_r.close();

k = 0;
while (file_conc >> darg && k < M)
{
    hc2[k++] = darg;
}
file_conc.close();

k = 0;
while (file_left >> iarg && k < N * M)
{
    hleft2[k++] = iarg;
}
file_left.close();

k = 0;
while (file_right >> iarg && k < N * M)
{
    hright2[k++] = iarg;
}
file_right.close();

input << "Method: Runge-Kutta 5, shared memory + block reduction" << endl;
input << "LOOP = " << LOOP << endl;
input << "Reactions: " << N << endl;
input << "Reagents/species: " << M << endl;
input << "Time step = " << dt << endl;

int* dleft = nullptr;
int* dright = nullptr;
double* dc = nullptr;
double* dccc = nullptr;
double* dk_f = nullptr;
double* dk_r = nullptr;
double* dk1 = nullptr;
double* dk2 = nullptr;
double* dk3 = nullptr;
double* dk4 = nullptr;
double* dk5 = nullptr;

checkCuda(cudaMalloc((void**)&dk_f, N * sizeof(double)), "cudaMalloc dk_f");
checkCuda(cudaMalloc((void**)&dk_r, N * sizeof(double)), "cudaMalloc dk_r");
checkCuda(cudaMalloc((void**)&dleft, N * M * sizeof(int)), "cudaMalloc dleft");
checkCuda(cudaMalloc((void**)&dright, N * M * sizeof(int)), "cudaMalloc dright");
checkCuda(cudaMalloc((void**)&dc, M * sizeof(double)), "cudaMalloc dc");
checkCuda(cudaMalloc((void**)&dccc, M * sizeof(double)), "cudaMalloc dccc");
checkCuda(cudaMalloc((void**)&dk1, M * sizeof(double)), "cudaMalloc dk1");
checkCuda(cudaMalloc((void**)&dk2, M * sizeof(double)), "cudaMalloc dk2");

```

```

checkCuda(cudaMalloc((void**)&dk3, M * sizeof(double)), "cudaMalloc dk3");
checkCuda(cudaMalloc((void**)&dk4, M * sizeof(double)), "cudaMalloc dk4");
checkCuda(cudaMalloc((void**)&dk5, M * sizeof(double)), "cudaMalloc dk5");

checkCuda(cudaMemcpy(dk_f, hk_f2, N * sizeof(double), cudaMemcpyHostToDevice),
"cudaMemcpy dk_f");
checkCuda(cudaMemcpy(dk_r, hk_r2, N * sizeof(double), cudaMemcpyHostToDevice),
"cudaMemcpy dk_r");
checkCuda(cudaMemcpy(dleft, hleft2, N * M * sizeof(int), cudaMemcpyHostToDevice),
"cudaMemcpy dleft");
checkCuda(cudaMemcpy(dright, hright2, N * M * sizeof(int), cudaMemcpyHostToDevice),
"cudaMemcpy dright");
checkCuda(cudaMemcpy(dc, hc2, M * sizeof(double), cudaMemcpyHostToDevice),
"cudaMemcpy dc");
checkCuda(cudaMemcpy(dccc, hc2, M * sizeof(double), cudaMemcpyHostToDevice),
"cudaMemcpy dccc");

checkCuda(cudaMemset(dk1, 0, M * sizeof(double)), "cudaMemset dk1");
checkCuda(cudaMemset(dk2, 0, M * sizeof(double)), "cudaMemset dk2");
checkCuda(cudaMemset(dk3, 0, M * sizeof(double)), "cudaMemset dk3");
checkCuda(cudaMemset(dk4, 0, M * sizeof(double)), "cudaMemset dk4");
checkCuda(cudaMemset(dk5, 0, M * sizeof(double)), "cudaMemset dk5");

cudaEvent_t start_dev, stop_dev;
checkCuda(cudaEventCreate(&start_dev), "cudaEventCreate start");
checkCuda(cudaEventCreate(&stop_dev), "cudaEventCreate stop");

const int THREADS = 1024;
dim3 BLOCK_SIZE(THREADS);
dim3 GRID_SIZE((N + THREADS - 1) / THREADS);

const int THREADS_COPY = 256;
dim3 BLOCK_COPY(THREADS_COPY);
dim3 GRID_COPY((M + THREADS_COPY - 1) / THREADS_COPY);

// shared memory көлемі: концентрациялар + редукция массиві
size_t sharedBytes = M * sizeof(double) + THREADS * sizeof(double);

input << "CUDA block size = " << THREADS << endl;
input << "CUDA grid size = " << GRID_SIZE.x << endl;
input << "Dynamic shared memory per block = " << sharedBytes << " bytes" << endl;

if (sharedBytes > deviceProp.sharedMemPerBlock)
{
    cerr << "ERROR: required shared memory per block is too large: "
        << sharedBytes << " bytes, device limit: "
        << deviceProp.sharedMemPerBlock << " bytes" << endl;
    return -1;
}

float gpuTime = 0.0f;
checkCuda(cudaEventRecord(start_dev, 0), "cudaEventRecord start");

```

```

int step = 1;
do
{
    for (int jj = 0; jj < M; jj++)
    {
        // ----- Stage 1 -----
        clear_stage_value<<<1, 1>>>(dk1, jj);
        checkLastKernel("clear_stage_value dk1");

        cudaRungeCutta_stage_shared<<<GRID_SIZE, BLOCK_SIZE, sharedBytes>>>(
            1, N, M, dt, dleft, dright, dc, dk_f, dk_r,
            dk1, dk2, dk3, dk4, dk5, jj);
        checkLastKernel("cudaRungeCutta_stage_shared stage 1");

        // ----- Stage 2 -----
        clear_stage_value<<<1, 1>>>(dk2, jj);
        checkLastKernel("clear_stage_value dk2");

        cudaRungeCutta_stage_shared<<<GRID_SIZE, BLOCK_SIZE, sharedBytes>>>(
            2, N, M, dt, dleft, dright, dc, dk_f, dk_r,
            dk1, dk2, dk3, dk4, dk5, jj);
        checkLastKernel("cudaRungeCutta_stage_shared stage 2");

        update_after_stage_2<<<1, 1>>>(dc, dk1, N, jj);
        checkLastKernel("update_after_stage_2");

        // ----- Stage 3 -----
        clear_stage_value<<<1, 1>>>(dk3, jj);
        checkLastKernel("clear_stage_value dk3");

        cudaRungeCutta_stage_shared<<<GRID_SIZE, BLOCK_SIZE, sharedBytes>>>(
            3, N, M, dt, dleft, dright, dc, dk_f, dk_r,
            dk1, dk2, dk3, dk4, dk5, jj);
        checkLastKernel("cudaRungeCutta_stage_shared stage 3");

        update_after_stage_3<<<1, 1>>>(dc, dk1, dk2, N, jj);
        checkLastKernel("update_after_stage_3");

        // ----- Stage 4 -----
        clear_stage_value<<<1, 1>>>(dk4, jj);
        checkLastKernel("clear_stage_value dk4");

        cudaRungeCutta_stage_shared<<<GRID_SIZE, BLOCK_SIZE, sharedBytes>>>(
            4, N, M, dt, dleft, dright, dc, dk_f, dk_r,
            dk1, dk2, dk3, dk4, dk5, jj);
        checkLastKernel("cudaRungeCutta_stage_shared stage 4");

        update_after_stage_4<<<1, 1>>>(dc, dk1, dk3, N, jj);
        checkLastKernel("update_after_stage_4");

        // ----- Stage 5 -----

```

```

clear_stage_value<<<1, 1>>>(dk5, jj);
checkLastKernel("clear_stage_value dk5");

cudaRungeCutta_stage_shared<<<GRID_SIZE, BLOCK_SIZE, sharedBytes>>>(
    5, N, M, dt, dleft, dright, dc, dk_f, dk_r,
    dk1, dk2, dk3, dk4, dk5, jj);
checkLastKernel("cudaRungeCutta_stage_shared stage 5");

update_after_stage_5<<<1, 1>>>(dc, dccc, dk1, dk4, dk5, dk3, N, jj);
checkLastKernel("update_after_stage_5");
}

cudaRungeCutta_new_shared<<<GRID_COPY, BLOCK_COPY>>>(dc, dccc, M);
checkLastKernel("cudaRungeCutta_new_shared");

checkCuda(cudaMemcpy(hc2, dccc, M * sizeof(double), cudaMemcpyDeviceToHost),
    "cudaMemcpy dccc to host");

result << step << " ";
for (int i = 0; i < M; i++)
{
    result << setprecision(8) << hc2[i] << " ";
}
result << endl;

step++;
} while (step <= LOOP);

checkCuda(cudaEventRecord(stop_dev, 0), "cudaEventRecord stop");
checkCuda(cudaEventSynchronize(stop_dev), "cudaEventSynchronize stop");
checkCuda(cudaEventElapsedTime(&gpuTime, start_dev, stop_dev), "cudaEventElapsedTime");

result.close();

input << "GPU calc. time, shared memory version: "
    << setw(6) << gpuTime << " msec" << endl;
input.close();

checkCuda(cudaEventDestroy(start_dev), "cudaEventDestroy start");
checkCuda(cudaEventDestroy(stop_dev), "cudaEventDestroy stop");

cudaFree(dleft);
cudaFree(dright);
cudaFree(dk_f);
cudaFree(dk_r);
cudaFree(dc);
cudaFree(dccc);
cudaFree(dk1);
cudaFree(dk2);
cudaFree(dk3);
cudaFree(dk4);
cudaFree(dk5);

```

```
delete[] hc2;
delete[] hk_f2;
delete[] hk_r2;
delete[] hleft2;
delete[] hright2;

cudaDeviceReset();

return 0;
}
```

ҚОСЫМША В.

Қосымшаға авторлық куәлік

ҚАЗАҚСТАН РЕСПУБЛИКАСЫ

РЕСПУБЛИКА КАЗАХСТАН



СВИДЕТЕЛЬСТВО
О ВНЕСЕНИИ СВЕДЕНИЙ В ГОСУДАРСТВЕННЫЙ РЕЕСТР
ПРАВ НА ОБЪЕКТЫ, ОХРАНЯЕМЫЕ АВТОРСКИМ ПРАВОМ
№ 13255 от «17» ноября 2020 года

Фамилия, имя, отчество, (если оно указано в документе, удостоверяющем личность) автора (ов):
САРСЕМБАЕВ МАГЖАН СМАГУЛОВИЧ, УРМАШЕВ БАЙДАУЛЕТ АМАНТАЕВИЧ, ТЕМИРБЕКОВ
АТМАС НУРІАНОВИЧ, АМАНТАЕВА АЙДАНА БАЙТӨУЛЕТКЫЗЫ, АЙТМУКАШ ДАНИЯР
БОЛАТУЛЫ

Вид объекта авторского права: программа для ЭВМ

Название объекта: Информационно-вычислительная система для автоматизированной генерации
химических моделей описывающих горение реальных топлив

Дата создания объекта: 25.09.2020





Күжат түпнұсқасының <http://www.kazpatent.kz/rz> сайтының
"Авторлық құқық" бөлімінде тексеруге болады. <https://copyright.kazpatent.kz>

Подлинность документа возможно проверить на сайте kazpatent.kz
в разделе «Авторское право» <https://copyright.kazpatent.kz>

Подписано ЭЦП

Оспанов Е.К.