# پردازش مشترک پرس و جو در کش برای معماریهای CPU-GPU جفت شده


## چکیده

اخیراً، تعدادی طرح های پردازنده ظهور کرده است که در آنها CPU و GPU (واحد پردازش گرافیکی) در یک تراشه یکپارچه شده و سطح کش آخرین (LLC) را به اشتراک می گذارند. اما، پهنای باند حافظه اصلی چنین معماری های CPU-GPU جفت شده ای پائینتر ازGPU گسسته است. در نتیجه، پارادایم های (الگو) پردازش مشترک پرس و جوی GPU فعلی، شدیداً از توقف حافظه آسیب می بینند. در این مقاله، پارادایم جدید پردازش مشترک پرس و جو در کش برای پایگاههای داده پردازش تحلیلی آنلاین حافظه اصلی (OLAP) برروی معماریهای CPU-GPU جفت شده را پیشنهاد می کنیم. مخصوصاً، از پیش واکشی به کمک CPU برای به حداقل رساندن از دست رفتن اطلاعات کش در پردازش مشترک پرس و جوی GPU و از غیر فشرده سازی به کمک CPU برای بهبود عملکرد اجرای پرس و جو استفاده می کنیم. به علاوه، یک مکانیسم تطبیق هدایت شده با مدل هزینه برای توزیع حجم کار پیش واکشی، غیر فشرده سازی، و اجرای پرس و جو بین CPU و GPU، توسعه می دهیم. سپس نمونه اولیه سیستم را پیاده و آن را برروی دو A8 AMD APU و A10 جدید ارزیابی می کنیم. نتایج آزمایش نشان می دهد که 1) پردازش مشترک پرس و جو در کش، عملکرد پارادایم پردازش مشترک GPU پیشرفته را برروی A8 و A10 به ترتیب تا 30 و 33 درصد بهبود می بخشد؛ و 2) مکانیسم تطبیق توزیع حجم کار، عملکرد پرس و جو برروی A8 و A10 را به ترتیب 36 و 40 درصد بهبود می بخشد.


## 1. مقدمه

پارادایم پردازش مشترک پرس و جو برروی GPU، وسیله ای موثر جهت بهبود عملکرد پایگاههای داده حافظه اصلی برای OLAP است. در حال حاضر، اکثر سیستم ها بر مبنای معماریهای CPU-GPU هستند، که CPU و GPU ازطریق باس PCI-e نسبتاً آهسته ای به هم متصل شده اند. اخیراً، طرح هایی از پردازنده ظهور

کرده است که در آنها CPU و GPU در یک تراشه باهم یکپارچه شده و LLC را به اشتراک می گذارند. به طور مثال، معماری واحد پردازش تسریع شده AMD (APU)، CPU و GPU را در یک تراشه یکپارچه کرده و اینتل، آخرین نسل پردازنده Ivy Bridge را در اواخر آوریل 2012، منتشر نمود. در مورد معماریهای ناهمگن در حال ظهور ، سرعت پائین PCI-e ، مسئله مهمی به شمار نمی رود. برای مکانیسم های پردازش داده جدید، معماری CPU-GPU جفت شده فراخوانده می شود. مطالعاتی در رابطه با طرح های مشارکتی و دانه ریز برای پردازش مشترک جستجو و حجم کارهای پردازش داده های دیگر انجام شده است ( مثلاً انبارهای کلید- مقدار و MapReduce).

علی رغم اثربخشی مطالعات قبلی پیرامون پردازش مشترک پرس و جو روی معماریهای جفت شده، در مطالعات قبلی، CPU و GPU هر دو برروی حجم کارهای همگن اجرا شدند. اما، به خاطر طراحی معماری منحصر به فرد معماریهای CPU-GPU جفت شده، چنین طرح های توزیع حجم کار همگنی می توانند مانع عملکرد پردازش مشترک پرس و جو روی GPU شوند. از طرف دیگر، قدرت GPU در معماری جفت شده کمتر از معماری گسسته است و اینکه در معماری جفت شده GPU( معمولاً DDR3) به حافظه اصلی با پهنای باند بسیار پائین تر از حافظه GPU گسسته دسترسی دارد (معمولاً GDDR5). این دو عامل منجر به استفاده کم از GPU در معماری جفت شده به خاطر توقف حافظه می گردند. طراحی GPU ذاتی مدل اجرای یک برنامه چند داده (SPMD) و طبیعت درست هسته های GPU، موجب می شود GPU در معماری جفت شده نسبت به توقف های حافظه حساس تر ظاهر شود. در این مقاله، نحوه کاهش توقف حافظه که GPU از آن رنج می برد را مورد پژوهش قرار داده و عملکرد پردازش مشترک پرس و جو روی معماری CPU-GPU جفت شده را بهبود می بخشیم.

قابلیت محاسباتی GPU روی معماری های CPU-GPU جفت شده جدید، بالاتر از CPU است. به طور مثال، روی AMD APU A8 و A10، GFLOPS[1] GPU 5 یا 6 برابر بالاتر از CPU است. اگر داده های ورودی در کش موجود باشند، آنگاه قابلیت محاسباتی خام بسیار خوب GPU منجر به سرعت بخشیدن بسیار مشابه می گردد. اما، به خاطر تاثیر فوق الذکر توقف حافظه بر پردازش مشترک GPU، زمانی که داده ها نمی توانند

[1] . Giga Floating Point Operations per Second

خودشان را با کش وفق دهند، آنگاه سرعت بخشیدن در حد پائین یعنی 2 است ( در بخش 1. 3 جزئیات بیشتری راجع به این مسئله مطرح شده است). بنابراین، سئوال طبیعی این است که چگونه می توان مطمئن شد که برای رهایی و آزاد نمودن توان GPU، مجموعه پردازش مشترک پرس و جوی عملی می تواند خود را با کش اندازه کند (برازش).

در این مقاله، پارادایم جدید پردازش مشترک پرس و جودر کش برای پایگاههای داده حافظه اصلی روی معماریهای CPU-GPU جفت شده را پیشنهاد می کنیم. مخصوصاً، از پیش واکشی به کمک CPU، برای به حداقل رساندن از دست رفتن اطلاعات کش، و از طرح های غیر فشرده سازی به کمک GPU و CPU، برای بهبود عملکرد اجرای پرس و جو، استفاده می کنیم. مهم نیست غیر فشرده سازی درگیر باشد یا خیر، مهم این است که طرح پیشنهادی تضمین می کند که داده های ورودی برای پردازش مشترک پرس و جوی GPU، پیش واکشی شده اند. بنابراین، اجراهای GPU عمدتاً برروی داده های داخل کش انجام می شود، بدون اینکه از توقف حافظه آسیب ببیند. مخصوصاً، برخلاف توزیع همگن حجم کار در پارادایم های پردازش مشترک پرس و جوی قبلی، توزیع حجم کار فعلی، ناهمگن است: حال هسته CPU می تواند عملیات پیش واکشی حافظه، غیر فشرده سازی، و حتی پردازش پرس و جو را انجام دهد و GPU می تواند غیر فشرده سازی و پردازش پرس و جو را اجرا نماید. در اینجا یک مکانیسم تطبیق هدایت شده با مدل هزینه برای توزیع حجم کارپیش واکشی، غیر فشرده سازی و ارزیابی پرس و جو بین CPU و GPU نیز توسعه می دهیم. با شکافت وسیله، به تخصیص منابع دانه ریز دست می یابیم که CPU یا GPU را به واحدهای زمان بندی کوچکتری تقسیم می کند (با شیوه های بر مبنای نرم افزار یا زمان اجرای OpenCL).

دراینجا نمونه اولیه ای از سیستم را پیاده و آن را برروی دو AMD APU جدید، یعنی A8 و A10 ارزیابی می کنیم. نتایج آزمایش نشان می دهد که 1) پردازش مشترک پرس و جو در کش می تواند عملکرد پردازش مشترک پرس و جوی GPU را برروی A8 و A10 به ترتیب تا 30 و 33 درصد بهبود دهد و 2)مدل هزینه پیشنهادی می تواند توزیع حجم کار مناسب را پیش بینی نماید و مکانیسم های تطبیق توزیع پیشنهادی عملکرد پرس و جورا 40- 36 درصد بهبود می بخشند.

رئوس این مقاله به شرح ذیل است. در بخش 2، پیشینه و اصول مقدماتی معماریهای جفت شده و OpenCL ، در بخش 3، طراحی و پیاده سازی پردازش مشترک پرس و جو در کش و در بخش 4، مدل هزینه را معرفی می کنیم. در بخش 5، نتایج آزمایش را مطرح کرده، در بخش 6 کارهای مرتبط را مرور کرده و در بخش 7 نتایج مطرح می گردد.

## 2. اصول مقدماتی و پیشینه

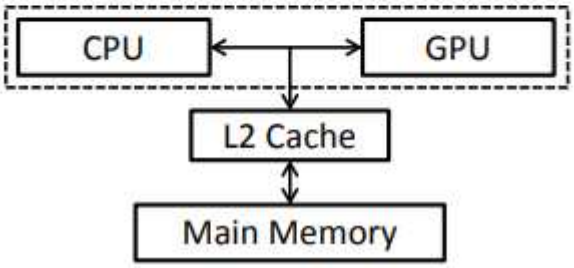
این بخش، به معرفی پیشینه و اصول مقدماتی معماریهای جفت شده و OpenCL می پردازد.

### 1. 2 معماریهای سیستم ناهمگن

در فیلد معماری کامپیوتر، طرح های معماری ناهمگنی در حال ظهور می باشند. محققین طرح های ناهمگن متفاوتی در پردازنده های مدرن/ آتی پیشنهاد کرده اند، که تلاش می کنند عملکرد را بهبود بخشیده، مصرف انرژی را کاهش داده یا هر دو را انجام دهند. این مقاله بر معماری CPU-GPU جفت شده تمرکز می کند.

طراحی معماری جفت شده در شکل 1 نشان داده شده است. CPU و GPU در تراشه یکسانی یکپارچه شده اند، که باس PCI-e را حذف می نماید. به علاوه، در این مطالعه، CPU و GPU کش L2 را به اشتراک می گذارند که امکان استفاده مجدد از داده ها بین آنها را فراهم می آورد. در AMD APU های فعلی، کلیه موارد دسترسی به داده ها از طریق پل شمالی یکپارچه (UNB) انجام می شوند که CPU، GPU و حافظه اصلی را به هم متصل می نماید. جدول 1 پیکره بندی های سخت افزاری دو نسل از AMD APU را نشان می دهد (به عبارتی A8-3870K و A10-7850K). برای مقایسه، پیکره بندی آخرین Radeon R9 270 را نیز به عنوان نمونه ای از GPU گسسته فهرست بندی می نماییم. در معماری جفت شده، به خاطر محدودیت های مساحت تراشه، GPU، در فرکانس ساعت (کلاک) پائین تر، دارای تعداد هسته بسیار کوچکتری است. در AMD APU های قبلی مثل A8 3870K، به اشتراک گذاشتن حافظه توسط بافر کپی صفر نسبتاً کوچکی انجام می شود. آخرین Kaveri APUs مثل A10-7850K از حافظه مجازی مشترک (SVM) پشتیبانی می کنند که به اشتراک گذاشتن حافظه را به کل فضای حافظه اصلی توسعه می دهند. پهنای باند حافظه نسبتاً پائین است (29.8GB/s)، زیرا DDR3 از لحاظ فضایی برای اپلیکیشن های حساس به تاخیر حافظه طراحی شده است. برای GPU های گسسته، GDDR5 پهنای باند 364GB/s فراهم می نماید.

| | A8 3870K | | A10 7850K | | Radeon R9 270 |
|---|---|---|---|---|---|
| Core type | CPU | GPU | CPU | GPU | GPU |
| # Cores | 4 | 400 | 4 | 512 | 1280 |
| Core frequency(MHz) | 3000 | 600 | 4000 | 720 | 925 |
| Shared memory (GB) | 0.5 | | 32(whole) | | N/A |
| Peak memory bandwidth (GB/s) | 5.6 | 24.5 | 7.8 | 28.9 | 179.2 |
| Cache size(MB) | 4 | | 4 | | N/A |




شکل 1

| Notation | Description |
|---|---|
| $XPU$ | CPU or GPU |
| $N$ | The input size (of D) in bytes |
| $\|C\|$ | The number of CPU CUs |
| $\|G\|$ | The number of GPU CUs |
| $B_{XPU}$ | The peak bandwidth (GB/s) of XPU to the shared main memory area |
| $S$ | The preset data size to be prefetched |
| $F$ | F $\in$ {P,D,E} |
| $T_F$ | The actual execution time of F |
| $comp_F^{XPU}$ | The computation time of F on XPU |
| $M_F^{XPU}$ | Memory access time of F |
| $C_F, G_F$ | The number of CPU and GPU CUs assigned to F |
| $\#I_{XPU}^F$ | The number of instructions of F on XPU |
| $cr$ | Compression ratio |
| $L_M^{XPU}$ | The access latency between XPU and main memory |
| $L_C^{XPU}$ | The access latency between XPU and L2 cache |
| $R_F$ | The output throughput of F in bytes |
| $IPC_{XPU}$ | The theoretical peak instruction per cycle on XPU |

جدول 1

در حقیقت، طراحی معماری سفارشی پیاده شده در PlayStation 4 از GDDR5 به عنوان حافظه اصلی، عمدتاً برای بهبود عملکرد گرافیک استفاده می نماید. در این مقاله، بر معماریهای CPU-GPU جفت شده با استفاده از DDR3 به عنوان حافظه اصلی تمرکز می کنیم، که در بازار کالای تجاری فعلی معمولتر هستند.

### 2. 2 واسط برنامه نویسی یکپارچه

زبان محاسباتی باز (OPenCL) یک زبان برنامه نویسی یکپارچه برای معماریهای ناهمگن است. برنامه های OpenCL را می توان یک مرتبه کدگذاری و برروی وسایل سازگار با OpenCL اجرا نمود. مطالعات موجود نشان داده است که برنامه ها در OpenCL می توانند به عملکردی نزدیک به زبان های مختص پلتفرم نظیر

CUDA برای NVIDIA GPU و OpenMP برای CPU دست یابند. به طور مثال، Fang وهمکاران [11] توضیح می دهند که پیاده سازیهای بر مبنای CUDA حداکثر 30 درصد بهتر از پیاده سازیهای بر مبنای OpenCL برروی NVIDIA GPU هستند. در بسیاری از سناریوها، OpenCL برروی CPU، حتی بهتر از OpenMP عمل می کنند.

کلیه وسایل سازگار با OpenCL با معماری منطقی یکسانی به نام compute device (وسیله محاسباتی)، نگاشته می شوند. هر وسیله محاسباتی از تعدادی واحدهای محاسباتی (CU) تشکیل می شود. به علاوه، هر CU محتوی عناصر پردازش متعددی است که به سبک SPMD اجرا می شوند. برروی APU، CPU و GPU به صورت دو وسیله محاسباتی برنامه نویسی می شوند. هر هسته CPU به صورت یک CU نگاشته شده و هر GPU CU معادل یک چندپردازنده است. تکه کد اجرا شده توسط یک وسیله خاص، کرنل نامیده می شود. کرنل از گروههای کاری متعددی برای اجرا استفاده کرده و هر گروه کاری، محتوی تعدادی آیتم های کاری است. گروه کاری با CU نگاشته شده (طرح ریزی) و آیتم های کاری متعدد به طور همزمان برروی CU اجرا می شوند. اجرای گروه کاری برروی معماری هدف، مختص فروشنده است. به طور مثال، AMD معمولاً 64 آیتم کاری را در جبهه موج و NVIDIA 32 آیتم کاری را در وارپ اجرا می کند. در این مقاله از واژگان فنی AMD استفاده می کنیم. کلیه آیتم های کاری در جبهه موج یکسان به شیوه یک دستورچندداده (SIMD) اجرا می شوند.

یکی از ویژگیهای جدید OpenCL برای پشتیبانی از تقسیم یک وسیله به زیروسایل متعدد، شکافت وسیله است. با این ویژگی، دو وظیفه متفاوت می توانند به طور همزمان برروی یک وسیله اجرا شوند. بنابراین، منابع سخت افزاری برروی یک وسیله در میان وظایف متعدد به اشتراک گذاشته شده و بدین طریق تخصیص منابع دانه ریز حاصل می گردد. در حال حاضر، از شکافت وسیله برروی اکثر وسایل CPU سازگار با OpenCL به طور کامل پشتیبانی می شود. اما، OpenCL، از شکافت وسیله برروی وسایل GPU پشتیبانی نمی کند. بنابراین، یک شیوه بر مبنای سخت افزار برای تقلید از شکافت وسیله برروی GPU توسعه می دهیم (زیربخش 3. 2. 3).

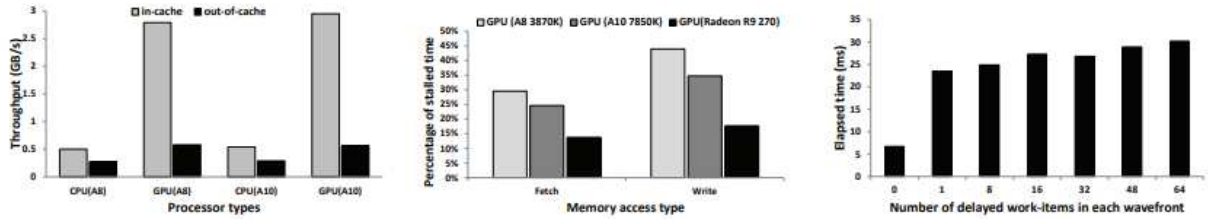
## 3. طراحی و پیاده سازی

در این بخش، ابتدا انگیزه توسعه پارادایم پردازش مشترک پرس و جو در کش را مطرح می کنیم. سپس به روش آزمایشی عملکرد حافظه GPU برروی دو AMD APU جدید را مورد ارزیابی قرار می دهیم. در بخش 5، راه اندازی آزمایشی با جزئیات مطرح شده است. اصولاً، سناریوی زیررا در نظر می گیریم: کلیه روابط و شاخص های پایگاههای داده در حافظه اصلی ذخیره می شوند. همانند مطالعات قبل، پرس و جوها ( عمدتاً پرس و جوهای OLAP) را می توان به صورت جزئی یا کلی در CPU یا GPU اجرا نمود. هدف ما از این کار بهبود کارایی پردازش مشترک پرس و جوی یک پرس و جو برروی معماری جفت شده است، مثل مطالعات قبل در مورد پردازش مشترک پرس و جو. در اینجا آزمایشات تحریک کننده و انگیزه بخشی انجام می دهیم که عملیات های پایه در پایگاههای داده برروی دو APU اجرا می شوند. برروی هر دو پلتفرم، به تعدادی مشاهدات مشترک دست یافته ایم که انگیزه پردازش مشترک پرس و جو در کش برروی معماری جفت شده را فراهم می آورند. سپس، طراحی و پیاده سازی مفصل پارادایم پردازش مشترک پرس و جوی پیشنهادی را مطرح می کنیم. مدل هزینه هدایت و راهنمایی تطبیق حجم کار در بخش 4 مطرح شده است.

### 1. 3 انگیزه ها

مشاهدات زیر انگیزه طراحی پردازش مشترک پرس و جو در کش را فراهم آورده اند.

مشاهده 1: عملکرد پردازش در کش GPU بسیار بالاتر از CPU است. شکل 2a مزایای حاصله از کش برای CPU و GPU را نشان می دهد. آزمایش، کارایی اجرای اسکن های متوالی ساده زیاد برروی رابطه یکسان را اندازه گیری می کند ( در اینجا 100 اسکن را اجرا می کنیم، اما تاثیر از دست دادن اجباری در اسکن اول را حذف می نماییم). ابتداعاً رابطه در حافظه ذخیره می شود. موارد درون کش و بیرون کش، جداولی با اندازه های 1MB و 16MB مطرح می کنند. در مقایسه با CPU،زمانی که اندازه رابطه از اندازه کش L2 تجاوز می کند، آنگاه GPU پرش تندتری به معرض نمایش می گذارد. بنابراین، GPU مجدداً از کش به مزایای عملکردی بیشتری نسبت به CPU دست می یابد، به شرطی که داده ها در کش L2 موجود باشند.




(a) In-cache processing benefit (b) Memory stalls on GPUs (c) Wavefront efficiency

شکل 2

مشاهده 2: پهنای باند حافظه GPU در معماری جفت شده بسیارپائین تر از GPU در معماری گسسته است. شکل 2b درصد زمان توقف حافظه را نشان می دهد که از پروفایلر AMD CodeXL بدست آمد، زمانی که اسکن جدول برروی GPU جفت شده و GPU گسسته در جدول 1 اجرا گردید. در سیستم جفت شده، پهنای باند حافظه محدود تر و دسترسی به حافظه بیشتر متوقف شده است. GPU در معماری جفت شده، نسبت به معماری گسسته، بیشتر مقید به حافظه است. نتیجه بدست آمده به ما انگیزه یافتن راهی تهاجمی تر برای کاهش اندازه دسترسی به داده ها از حافظه اصلی برای GPU را می دهد.

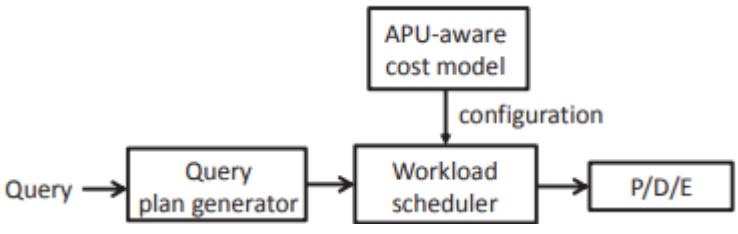
مشاهده 3: مدل اجرای SPMD و طراحی هسته درست و منظم GPU، می تواند عملکرد پردازش مشترک پرس و جوی GPU را شدیداً تحلیل دهد. کلیه آیتم های کاری در گروه کاری، در جبهه موج و به شیوه قفل به گام برروی GPU، گروه بندی و اجرا می شوند. حتی اگر فقط یک آیتم کاری با دسترسی به حافظه به تاخیر بیفتد، کل گروه کاری به تاخیر می افتد. شکل 2c عملکرد اسکن (پیمایش) جدول با دسترسی های تصادفی بر روی GPU را نشان می دهد، زمانی که تعداد آیتم های کاری به تاخیر افتاده به خاطر از دست دادن اطلاعات کش L2 از صفر به 64 افزایش می یابد. صفر به این معناست که همه داده های ورودی درکش قرار دارند و 64 به این معناست که کلیه آیتم های کاری در گروه کاری دارای اطلاعات از دست رفته در کش هستند. زمانی که هیچ آیتم کاری به تاخیر نمی افتد، آنگاه زمان سپری شده کاملاً کوتاه است. اما، مادامی که آیتم های کاری به تاخیر افتاده وجود دارد، عملکردبه تندی تحلیل می رود.

این مشاهدات مغایر با پارادایم های پردازش مشترک پرس و جوی موجود هستند. پارادایم پردازش مشترک پرس و جوی پیشرفته و سایر پارادایم های پردازش مشترک داده های مشابه برروی معماریهای CPU-GPU جفت شده، نمی توانند از آن ویژگیها بهره ببرند. به ویژه، کلیه مطالعات قبلی حجم کارهای همگنی را به CPU و

GPU تخصیص می دهند. طبق این مشاهدات، توقف حافظه موج تحلیل شدید GPU می شود، که معمولاً عامل اصلی عملکرد برای پایگاههای داده به حساب می آید. علی رغم بهبود دانه ریز و مشارکتی در مطالعات قبل، توزیع همگن حجم کارموجب توقف زیاد حافظه برروی GPU می شود. تحلیل عملکرد ناشی از توقف حافظه برروی GPU بسیار شدیدتر از CPU است. عملکرد پردازش مشترک پرس و جوی ایده آل باید تا حد امکان از محاسن GPU استفاده نماید (به عبارتی عملکرد بسیار بالاتر پردازش داده ها در کش).

### 2. 3 طراحی و پیاده سازی

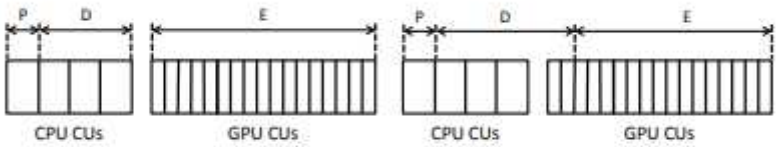
در اینجا با توسعه پردازنده پرس وجو برمبنای OpenCL، OmniDB، یک پردازنده مشترک پرس و جودر کش، طراحی و توسعه می دهیم. این سیستم برای پشتیبانی از OLAP طراحی شده و بر پرس و جوهای فقط خواندنی تمرکز می کند. سیستم مذکور از به روزرسانی های آنلاین پشتیبانی نمی کند. در عوض، رابطه ای برای به روزرسانی های دسته ای می سازد. شکل 3 یک نمای کلی از معماری را نشان می دهد که مخصوصاً برای معماریهای CPU-GPU جفت شده طراحی شده است ( این مطالعه بر AMD APU تمرکز می کند). پرس و جوها توسط یک مولد (ژنراتور) برنامه پرس و جو با استفاده از بهینه ساز سبک Selinger پردازش می شوند. مدل هزینه آگاه از APU به ویژگیهای معماری جفت شده دست یافته و برنامه تخصیص حجم کار و زمان اجرای پیش بینی شده برای پرس و جو را تولید می نماید. در اینجا سه ماژول فانکشنال یا کاربردی مشترک در پارادایم پردازش پرس و جو در کش را به طور خلاصه مطرح می کنیم: پیش واکشی (P)، غیر فشرده سازی (D، اختیاری)، و اجرای حقیقی پرس و جو (E). هر CU می تواند برروی هر واحد P/D/E کار کند. ماژول های تابعی مذکور توسط زمان بند حجم کار برای CU های موجود زمان بندی می شوند. پارادایم پردازش مشترک پرس و جو در کش پیشنهادی، در پایگاههای داده با یا بدون فشرده سازی کاربرد دارد. اگر داده ها فشرده شوند، آنگاه قبل از اجرای پرس و جو به عملیات غیر فشرده سازی نیاز می باشد. و اگر فشرده نشوند، آنگاه داده ها با اجرای پرس و جو مستقیماً پردازش می شوند.



شکل 3

طبق مطالعات قبل، از پیش واکشی برای مخفی کردن یا پنهان کردن تاخیر حافظه در داخل عملگرهای پایگاه داده استفاده می شود.

بسته به نحوه تخصیص ماژول های تابعی به کلیه CU های موجود، چهار پیکره بندی اجرای متفاوت وجود دارد که در شکل 4 به تصویر کشیده شده اند.




(c) cPDE-b (d) cPDE-g

شکل 4

در شکل 4a (PE)، تقریباً کل CU ها برروی CPU و GPU به اجراهای پرس و جو تخصیص داده شده، و فقط یک CPU CU برای انجام عمل واکشی باقی می ماند. این مسئله برای سناریوهایی مناسب است که داده ها بدون فشرده سازی ذخیره می شوند، یا اجرای پرس و جو مستقیماً برروی داده های فشرده شده انجام می شود. در صورت نیاز به غیر فشرده سازی ، سه پیکره بندی ممکن برای اجرا به نامهای cPDE-c, cPDE-b و cPDE-g وجود دارد. در اینجا *DE boundary* به تقسیم موقعیت بین ماژول های تابعی D و E گفته می شود. مقدار مرز DE، تعداد CU های تخصیص داده شده به D را نشان می دهد که از سمت CPU به سمت GPU جابجا می شوند. شکل 4b موردی را نشان می دهد که مرز DE برروی CPU قرار دارد، در حالیکه شکل 4c و 4d نشان می دهند که مرز DE درست برروی مرز بین CPU و GPU و برروی GPU قرار دارد. به علاوه، روش پیشرفته ای برای دستیابی به توزیع حجم کار دانه ریز پیاده می کنیم، زمانی که E بین دو وسیله قرار دارد. مخصوصاً، اجرای پرس و جو را می توان به گام هایی تقسیم نمود ( یک گام به یک عملگر در پرس و جو یا پردازش دانه ریزتر در یک عملگر گفته می شود). از آنجایی که پرس و جو برر وی تعدادی CPU CU و کل GPU در PE و cPDE-c اجرا می شود، در نتیجه هر گام را می توان برروی دو وسیله با حجم کار متفاوت زمان بندی نمود و بدین طریق به عملکردی متعادل و بهینه در دو سمت دست یافت.

در این مقاله، برای توضیح، از دو عملیات پایگاه داده مشترک و بنیادی به عنوان مثال استفاده می کنیم ( به عبارتی انتخاب و اتصال هش). بدون شاخص، انتخاب با استفاده از تابع فیلتراولیه پیاده می شود که مسند به عنوان تابع فیلتر عمل می کند. اتصال هش، عملیاتی کاملاً پیچیده است. حتی بعد از بهینه سازیهای مختلف حافظه، توقف حافظه می تواند جداً به عملکرد اتصال آسیب وارد نماید. در اینجا از روش اتصال هش پیشرفته استفاده کردیم. هر فاز از اتصال هش (پارتیشن، ساخت، پروب) به مراحل و گام هایی تقسیم می شود. در ادامه جزئیات پیاده سازی هر مولفه را مطرح می کنیم.

### 1. 2. 3 پیش واکشی

مطالعات قبل راجع به اثربخشی پیش واکشی حافظه در پنهان کردن توقف حافظه با محاسبات مفید در پایگاههای داده توضیح داده اند. در زمینه پردازش مشترک پرس و جو در کش برروی معماری جفت شده، مجدداً به تاثیر پیش واکشی با توجه خاص به چهار پیکره بندی اجرا نگاه می کنیم.

ساختار پیش واکشی پیشنهادی بر مبنای تکنیک پیش واکشی پیشنهاد شده توسط Zhou و همکاران [39] بوده و در معماری موازی عظیمی مثل GPU کاربرد دارد. از ساختار WAS [۲] برای ذخیره موقتی داده های پیش واکشی شده استفاده شده است. رشته های زیادی وجود دارد که به طور همزمان برروی GPU کار می کنند و در یک زمان موارد زیادی از دسترسی به حافظه منتشر می نمایند. بنابراین، دسترسی به حافظه را به شیوه دسته ای در WAS جایگذاری می کنیم. در صورت نیاز به غیر فشرده سازی ، از دو WAS برای تشکیل خط لوله اجرای دو جفت تولید کننده- مصرف کننده داده استفاده می کنیم: از یکی برای پیش واکشی داده ها از ورودی فشرده شده برای غیر فشرده سازی و از دیگری برای ذخیره داده های غیر فشرده به عنوان ورودی اجراهای پرس و جو استفاده می شود. یکی دیگر از موضوعات مهم آن است که اندازه WAS به عملکرد کلی کمک می کند. اگر اندازه خیلی کوچک باشد، رشته کمک کننده (کمک رسانی) زمان کافی برای بارگذاری خطوط کش درخواست شده ندارد. اگر اندازه خیلی بزرگ باشد، رشته کمک کننده داده های مفید را خارج می کند. به طور ایده آل، باید کوچکتر از ظرفیت کش باشد. از آنجایی که رشته اصلی دارای داده های ساکن در کش دیگری است که هنوز از آنها استفاده می شود، در نتیجه برای اجتناب از از دست دادن تعارض، مقدار آستانه باید

[2] . work-ahead set

پائین تر آورده شود. دو ماژول کاربردی دیگر یعنی D و E فقط در صورتی می توانند به داده ها دسترسی یابند که پیش واکشی شده باشند. بنابراین، کمتر احتمال دارد که از دست رفتن اطلاعات کش به آنها آسیب وارد نماید. این مکانیسم به عوامل زیادی نظیر نسبت فشرده سازی و CU های تخصیص داده شده بستگی دارد. در بخش 4، برای بررسی این مسئله، یک مدل تحلیلی مطرح می کنیم.

در اینجا از تکنیک های پیش واکشی به شیوه ای دانه ریزتر استفاده می کنیم. برای واکشی، عملگرها باید به گامهایی تقسیم شوند. از آنجایی که انتخاب فقط دارای یک گام به عنوان تعریف است، در نتیجه گام را می توان در امتداد بعد داده ها تعریف نمود.

مخصوصاً فرض می کنیم تعداد آیتم های کاری مشغول کار برروی انتخاب $NDRange$ است. عملیات های انجام شده برروی داده ها در دامنه صفر تا $NDRange\text{-}1$ به عنوان گام اول در نظر گرفته شده است. برای آیتم کاری، داده هایی که در مرحله بعدی باید واکشی شوند، در موقعیت $(i+NDRange)$ قرار دارند. تعریف گام ها مطالعه قبلی ما را دنبال می کند. موقعیت بعدی حافظه مورد استفاده، از گام فعلی بدست می آید.

در عمل به دو دلیل پیش واکشی را برروی یک CPU CU ثابت می کنیم. اولاً، جبهه موج GPU از الگوی اجرای SIMD استفاده می کند. بنابراین در صورت بلوکه شدن هر آیتم کاری، سایر آیتم های کاری موجود در آن جبهه موج باید منتظرآیتم بلوکه شده بمانند، که این امر موجب می گردد GPU در پیش واکشی مشتمل بر از دست رفتن اطلاعات زیاد در کش، ناکارآمد ظاهر شود. ثانیاً، از آنجایی که یک GPU CU، چندپردازنده است، در نتیجه استفاده از یک GPU CU در جریان پیش واکشی بی فایده است.

### 2. 2. 3 فشرده سازی داده ها

یک شیوه موثر برای بهبود پردازش مشترک پرس و جو برروی GPU، فشرده سازی پایگاه داده است. این شیوه می تواند استفاده از پهنای باند را افزایش و مشکل توقف حافظه APU را حل کند.

در اینجا نمونه الگوریتم های فشرده سازی معرفی شده در کار قبل، من جمله NS، NSV، DICT، RLE ، Delta، Scale و FOR را انتخاب می کنیم. NS، NSV، DICT و RLE طرح های فشرده سازی اصلی هستند که از آنها می توان به شکلی مستقل یا مستقلاً استفاده نمود، در صورتی که الگوریتم های دیگر کمکی به

حساب می آیند از این نظر که از آنها فقط در کنارطرح های اصلی می توان برای بهبود هر چه بیشتر نسبت فشرده سازی استفاده نمود. در این بخش، طرح های فشرده سازی مذکور را به طور خلاصه شرح می دهیم.

جزئیات بیشتر راجع به پیاده سازی در مطالعه قبل یافت می شود. NS و NSV در جریان نمایش بیت هر عنصر، صفرهای مقدم مهمترین بیت ها را حذف می کنند. RLE مقادیر هر اجرا را بر مبنای جفت (مقدار، طول اجرا) ذخیره شده در دو آرایه نشان می دهد، که هر یک از آنها را می توان بیشتر فشرده نمود. برای طرح های کمکی، Delta هر مقدار را بر مبنای اختلاف با مقدار موقعیت قبل، رمزگذاری می کند. مقدار اول جهت غیر فشرده سازی در کاتالوگ ذخیره می شود.

در مواردی که فرمت عدد صحیح برای اپلیکیشن به اندازه کافی دقیق است، Scale مقادیر ممیز شناور را به اعداد صحیح تبدیل می کند. FOR هر مقدار در ستون را برای یک آفست از مقدار پایه رمزگذاری می کند. مقدار پایه معمولاً به عنوان کوچکترین مقدار آن ستون انتخاب می شود.

دراینجا از برنامه ریز فشرده سازی بکاررفته در مرجع [12] برای دستیابی به کاندیدهای برنامه فشرده سازی بهینه استفاده می کنیم. موضوعات اصلی عبارتنداز: یکپارچه سازی پروفایل عملکرد APU در برنامه ریز، و بهره برداری از پیش واکشی نرم افزار و پردازش مشترک پرس و جو در کش برای برآورد هزینه. آنالیز هزینه توسط مدل هزینه انجام می شود.

هر چند در مواردی با یک الگوریتم فشرده سازی اصلی ( نظیر NS و NSV)، از غیر فشرده سازی می توان اجتناب نمود، اما برنامه فشرده سازی آبشاری (کاسکاد) تولید شده توسط برنامه ریز فشرده سازی اغلب مستلزم غیر فشرده سازی است (یا حداقل غیر فشرده سازی جزئی). برای بینش های کافی در مورد نحوه تاثیرگذاری غیر فشرده سازی بر عملکرد پردازش مشترک پرس و جو، هر دو مورد نیاز یا عدم نیاز به غیر فشرده سازی را مورد پژوهش قرار می دهیم. نتایج تفصیلی در بخش 5 مطرح شده است.

در صورت نیاز به غیر فشرده سازی، غیر فشرده سازی، یک بلوک داده فشرده شده را واکشی نموده و سپس آن را در فرمت مورد نیاز غیر فشرده می نماید. درکنار پیش واکشی، از دو بافر WAS استفاده می شود. هر آیتم کاری که برروی غیر فشرده سازی کار می کند، موقعیت داده های بعدی را در اولین بافر WAS جایگذاری نموده و سپس عملیات پیش واکشی را انجام می دهد. خروجی غیر فشرده سازی، در یک بافر میانی ذخیره می شود که

به عنوان ورودی اجرای بعدی عمل می کند. برای هماهنگ نمودن پیشرفت D و E، یک فلگ (پرچم) مشترک (به اشتراک گذاشته شده) تعیین می شود که نشان می دهد غیر فشرده سازی برروی بلوک داده فشرده شده خاصی خاتمه یافته و E می تواند شروع به پردازش داده های غیر فشرده نماید.

### 3. 2. 3 شکافت وسیله

همان گونه که در شکل 4 نشان داده شده است، دو ماژول کاربردی برروی یک وسیله اجرا می شوند. هر ماژول کاربردی از یک یا چند کرنل OpenCL تشکیل می شود. برای تقسیم یک وسیله به زیروسایل متعدد به شکافت وسیله نیاز می باشد. هر زیروسیله می تواند کرنل تعدادی از ماژول های تابعی را اجرا نماید. بنابراین، وسیله یکسانی میان ماژول های تابعی به اشتراک گذاشته می شود. در حال حاضر از شکافت وسیله برروی CPU به طور کامل پشتیبانی می شود، اما برروی GPU های فعلی از آن پشتیبانی نمی شود.

در اینجا از شیوه بر مبنای نرم افزار ساده اما موثر برای دستیابی به شکافت وسیله برروی GPU استفاده می کنیم. برای پشتیبانی از دو کرنل OpenCL که به طور همزمان برروی GPU اجرا می شوند، مجبوریم آنها را در یک کرنل ادغام کرده و سپس آن را برروی GPU راه اندازی نماییم. دو کرنل اصلی با دستور شرطی if-else از هم متمایز می شوند. الگوریتم 1 نحوه ادغام دو کرنل K1 و K2 در یک کرنل K را به تصویر می کشد. فرض کنید K در کل $NDRangeSize$ آیتم کاری را احضار کرده و فرامی خواند. از پارامتر تنظیم دقیق $NDRangeSize1$ برای تنظیم شکافت وسیله برروی GPU استفاده می شود، به گونه ای که آیتم های کاری $NDRangeSize1$ برای K1 و آیتم های کاری $(NDRangeSize - NDRangeSize1)$ برای K2 راه اندازی می شوند. کلیه شاخصها در K2 باید طبق اطلاعات بعد، به روزرسانی شوند. در این راستا تضمین می کنیم که $NDRangeSize1$ ضرب انتگرال اندازه گروه کاری است ( به عبارتی تعداد آیتم های کاری در آن گروه کاری). بنابراین، هیچ گونه واگرایی شاخه ای به کرنل ادغام شده تحمیل نمی شود، زیرا از ویژگی OpenCL بهره می برد که واحد زمان بندی حجم کار، یک گروه کاری در زمان اجرای OpenCL است.

**Algorithm 1** Software-based device fission between two OpenCL kernels K1 and K2 on the GPU.

```
K(NDRangeSize, NDRangeSize1)
{
/* index represents work item ID in K */
if index < NDRangeSize1; then
   Execute K1;
else
   if index < NDRangeSize then
      /* Update the index to make it start from 0 for K2*/
      index ← index − NDRangeSize1;
      Execute K2;
}
```

الگوریتم 1

## 4. مدل هزینه

انتخاب پیکره بندی بهینه برای تنظیم دقیق پارامترهای مختلف ، وظیفه ای مهم تلقی می گردد، به ویژه در OpenCL که وسایل محاسباتی ناهمگن را هدف قرار می دهد. در این بخش، یک مدل هزینه برای برآورد زمان اجرای پردازش مشترک پرس و جوی چهار پیکره بندی اجرا برروی معماری جفت شده توسعه داده و سپس از آن برای تعیین مقادیر مناسب جهت تنظیم دقیق پارامترها و نیل به پائین ترین زمان اجرای برآورد شده استفاده می نماییم.

با وجود تعدد کارهای موجود پیرامون ساخت مدل هزینه برای اپلیکیشن ها برروی CPU یا GPU، تکامل معماری APU چالش های جدیدی به ارمغان می آورد. اولاً، پارادایم (الگوی) پردازش مشترک مستلزم آن است که مدل پیشنهادی ویژگیهای مختلف دو پردازنده را در معماریهای ناهمگن مد نظر قرار دهد. ثانیاً، ماژول های تابعی به طور همزمان اجرا شده و D و E را می توان به طور همزمان برروی دو وسیله مستقر نمود، که این امر پیش بینی درست عملکرد را سخت و دشوار می نماید. ثالثاً، پیش واکشی، تعداد موارد از دست دادن اطلاعات کش را تغییر و غیر فشرده سازی، اندازه داده های تحت دسترسی هر ماژول تابعی را تغییر می دهد. کلیه این عوامل باید در برآورد مد نظرقرار داده شوند.

از آنجایی که OpenCL خلاصه ای برای کلیه وسایل سازگار با OpenCL فراهم نموده است، در نتیجه با CPUیا GPU به عنوان پردازنده ای با معماری یکسان رفتار می کنیم که بر اساس قابلیت محاسباتی و پهنای باند حافظه متمایز شده است. سپس هر وظیفه را مستقلاً برروی CPU و GPU نمایش داده و به هزینه برروی

یک CU دست می یابیم. به علاوه، هزینه های کل عملیات پایگاه داده را به دو مولفه اصلی تقسیم می کنیم: هزینه محاسبات و هزینه حافظه. هزینه محاسبات براساس پیک فرضی IPC (دستورات هر سیکل) و تعداد دستورات بدست می آید. هزینه حافظه، پیش واکشی و غیرفشرده سازی را مد نظر قرار می دهد. در بقیه این بخش، ابتدا مدل مجرد (انتزاعی) را مطرح کرده و سپس از دو عملگر (انتخاب و اتصال هش) به عنوان مثالهایی برای معرفی مدل مجرد استفاده می کنیم.

### 1. 4 مدل مجرد (انتزاعی)

در این بخش برای دستیابی به پردازش مشترک، چهار پارادایم PE، cPDE-c، cPDE-b و cPDE-g را مطرح کرده ایم که در شکل 4 نشان داده شده اند. در اینجا بر نحوه ساخت مدل هزینه برای مواردی تمرکز می کنیم که پیش واکشی و غیر فشرده سازی یکپارچه و ادغام شده اند.

سپس از چهار طرح کلیدی زیر برای یافتن پیکره بندی درست استفاده می کنیم. اولاً از آنجایی که اجرای پرس و جو (E) در روش دانه ریز بهینه می شود، در نتیجه عملگرها باید در گامهایی مرحله بندی شوند. تعیین نسبت های بهینه میان همه گام ها برای دستیابی به زمان کل بهینه سخت و دشوار است. بنابراین، از مدل هزینه He و همکاران [19] برای بررسی این مسئله استفاده کرده ایم. ثانیاً، اثر کش بایستی وارد شود. به طور ایده آل، کلیه داده های کاری D و E مستقیماً از کش دسترس پذیرهستند و فقط P از از دست رفتن اطلاعات کش آسیب می بیند، که با محاسبه E و E می تواند مخفی شود. اگر از دست رفتن اطلاعات کش در D یا E رخ دهد، آنگاه در مدل هزینه باید پنالتی یا جریمه در نظر گرفته شود. ثالثاً، شکافت وسیله، یک وسیله را به زیروسایل متعددی تقسیم می کند که وظایف مختلفی برروی آنها اجرا می شوند. در آن سناریو، زمانی که از مرزهای DE متفاوتی استفاده می شود، آنگاه مدل هزینه می تواند زمان اجرا را به درستی برآورد نماید. چهارم اینکه، ماژول های تابعی به شیوه خط لوله ای کار می کنند، که می تواند موجب تاخیر شود ( به عبارتی تخصیص CU نامناسب به P/D/E). برای اجرا، تاخیر زمانی رخ می دهد که خروجی D قادر به تامین به موقع ورودیE نباشد. در مواردی که داده ها از قبل به درستی در کش L2 بارگذاری نمی شوند، D و E به تاخیر می افتند.

جدول 2 علامتگذاری استفاده شده در مدل هزینه را فهرست بندی می نماید.

پیش واکشی بر روی CPU CU اجرا می شود. حجم کار اصلی CPU CU برروی دستورات واکشی حافظه است. اندازه داده پیش واکشی شده در جریان دستیابی به پیش واکشی با S نشان داده می شود. برای تضمین اجرای واقعی پیش واکشی، از محاسبات سبک وزن استفاده می شود. بنابراین، کارایی پیش واکشی را می توان بر مبنای پهنای باند و قابلیت محاسباتی CPU CU محاسبه نمود.

$$R_P = \frac{|S|}{\frac{|S|}{B_{CPU}} + \frac{I^P_{CPU}}{IPC_{CPU}}} \qquad (1)$$

زمان اجرای D را می توان در دو مولفه محاسبه نمود: زمان محاسبه و زمان دسترسی به حافظه.

برای زمان محاسبه، تعداد دستورات اجرا شده برروی وسیله (وسایل) با پروفایلر OpenCL نظیر CodeXL یا AMD APP Profiler را شمرده، و زمان محاسبه کل دستورات را طبق پیک فرضی IPC پردازنده برای هر الگوریتم غیر فشرده سازی محاسبه می کنیم. برای دستیابی به نسبت فشرده سازی بهینه برروی هر ستون، از برنامه های متفاوتی استفاده می شود. بنابراین، تعداد دستورات برای غیر فشرده سازی در میان ستونها ثابت نیست. به همین خاطر برنامه فشرده سازی را براساس مدل پیشنهاد شده توسط Fang و همکاران [12] انتخاب می کنیم. در این راستا برای دستیابی به تعداد دستورات غیر فشرده سازی هر ستون برروی هر پردازنده، یعنی، $I^D_{XPU}$ محک زنی را قبل از اجرای پرس و جو اجرا می کنیم. طبق تعداد CPU CU و GPU CU های تخصیص داده شده به غیر فشرده سازی، می توانیم مقدار زمان اجرای دستوربرروی هر وسیله را بدست بیاوریم. زمان اجرای کل غیر فشرده سازی به زمان اجرای طولانی تر دو وسیله بستگی دارد که در معادله 2 نشان داده شده است.

$$\begin{aligned} comp_D^{XPU} &= max(comp_D^{CPU}, comp_D^{GPU}) \\ &= max(\frac{\#I^D_{CPU} \times \frac{C_D}{|C|}}{IPC_{CPU}}, \frac{\#I^D_{GPU} \times \frac{G_D}{|G|}}{IPC_{GPU}}) \end{aligned} \qquad (2)$$

زمان دسترسی به حافظه از زمان دسترسی از حافظه اصلی و کش داده های L2 تشکیل می شود. اگر $\frac{N}{comp_D^{XPU}} > R_P$ ، آنگاه مقدار $(N - comp_D^{XPU} \times R_P)$ داده در بایت باید از حافظه اصلی دسترس پذیر باشد. بنابراین، دارای زمان دسترسی به حافظه زیر هستیم.

$$M_D^{XPU} = (N - comp_D^{XPU} \times R_P) \times L_M^{XPU} + (comp_D^{XPU} \times R_P) \times L_C^{XPU} \quad (3)$$

در غیر این صورت، داده های ورودی قبل از استفاده کاملاً در کش پیش واکشی شده و بدین طریق معادله 4 بدست می آید.

$$M_D^{XPU} = N \times L_C^{XPU}, \quad (4)$$

زمان اجرای حقیقی D از رابطه زیر بدست می آید:

$$T_D = comp_D^{XPU} + M_D^{XPU} \quad (5)$$

به همین نحو، E را برآورد می کنیم: زمان محاسبه $(comp_E^{XPU})$ را به صورت معادله 2 و زمان دسترسی به حافظه $(M_E^{XPU})$ را به صورت معادله 3 و 4 محاسبه می کنیم. جمع $comp_E^{XPU}$ و $M_E^{XPU}$ را به صورت زمان اجرای فرضی E $(T'_E)$ تعریف می کنیم، با این فرض که خروجی D قادر به تامین به موقع ورودی E است. اما، از آنجایی که در اجرای واقعی، D و E یک زنجیره تولیدکننده- مصرف کننده تشکیل می دهند، در نتیجه در صورت تخصیص CU نامناسب، E بایستی منتظر D بماند. تاخیر، از اختلاف بین $T_D$ و $T'_E$ بدست می آید. اگر $T_D \leq T'_E$، آنگاه D می تواند E را به موقع تغذیه کند. بنابراین، دارای برآورد زیر هستیم.

$$T_E = T'_E \quad (6)$$

در غیر این صورت، زمان پردازش E محدود به D می شود. بنابراین، دارای برآورد زیر هستیم.

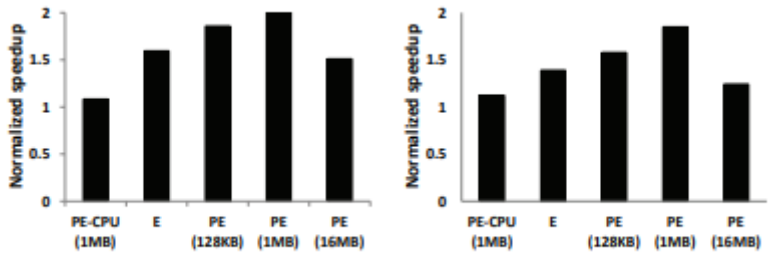
$$T_E = T_D \quad (7)$$

تعداد CU تخصیص داده شده به کلیه ماژول های تابعی باید در حدCU های موجود باشد. بنابراین، شرایط زیر نیز باید ارضاو تامین گردد.

$$0 \leq C_P + C_D + C_E \leq |C| \quad (8)$$

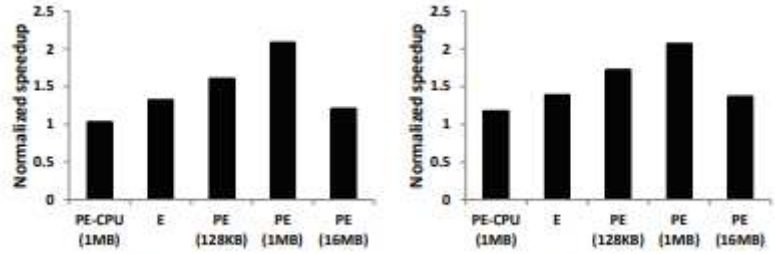
$$0 \leq G_D + G_E \leq |G| \quad (9)$$

هدف مدل هزینه پیشنهادی، یافتن برنامه بهینه جهت به حداقل رساندن زمان اجرای کل مثل معادله 10 است.



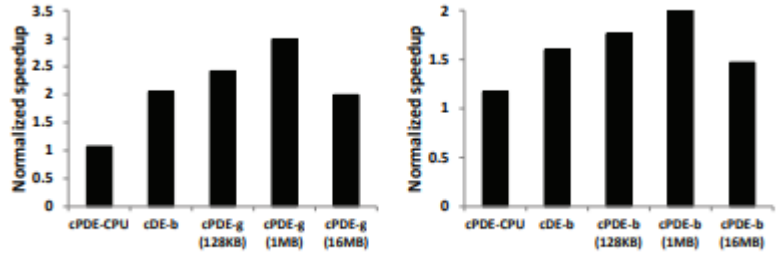

(a) Selection (b) Hash Join

شکل 5




(a) Selection (b) Hash Join

شکل 6

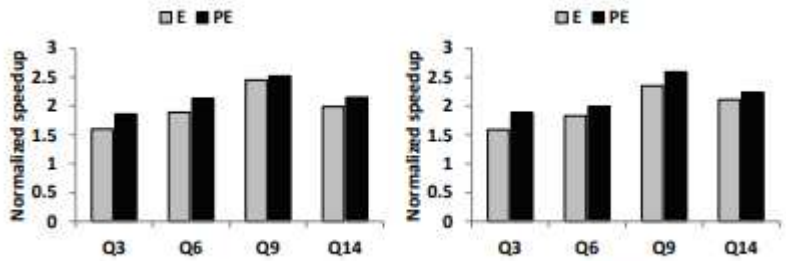



(a) Selection (b) Hash Join

شکل 7

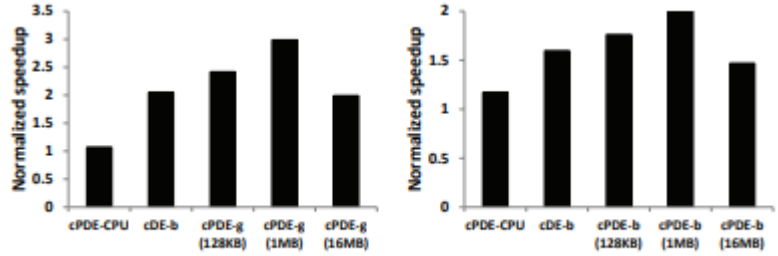
| Compression plan | Compression ratio | Decompression | Time (ms) |
|---|---|---|---|
| (A):RLE | 3.34% | No | 11.3 |
| (B):NS | 100% | No | 104.7 |
| (C):RLE,[$\epsilon$ \| NS] | 2.76% | No | 10.1 |
| (D):RLE,[[Delta, NS] \| NS] | 2.55% | Partial | 37.7 |
| (E):Delta, NS | 25% | Full | 145.4 |

جدول 3



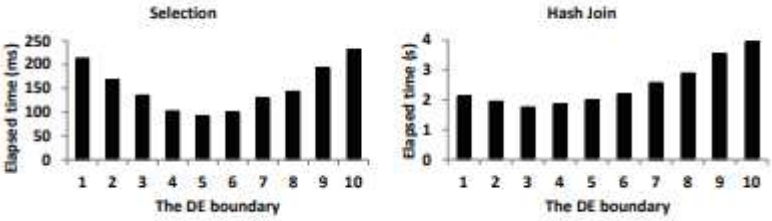

(a) A8 (b) A10

شکل 7



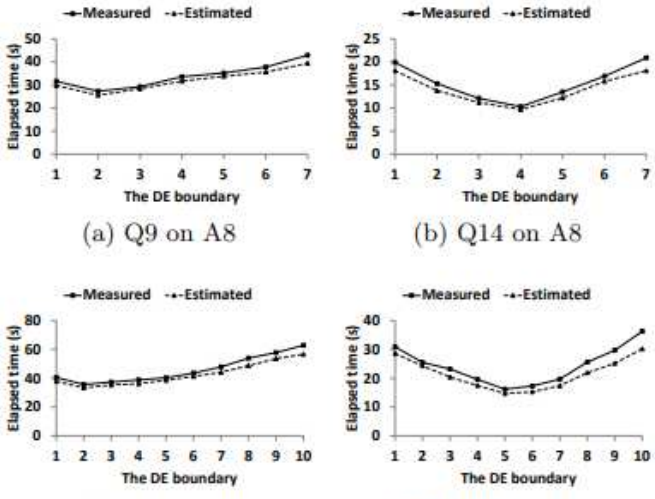

(a) Selection (b) Hash Join

شکل 8




(a) Selection (b) Hash Join

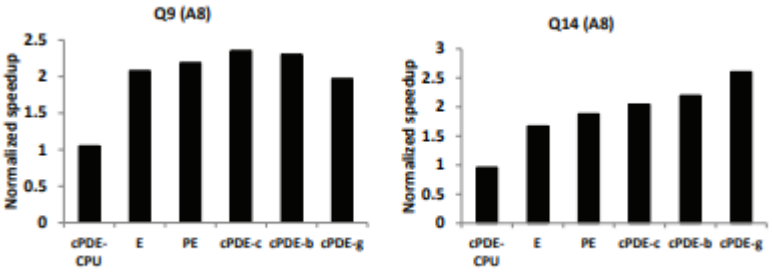
شکل 9




(a) Q9 on A8 (b) Q14 on A8

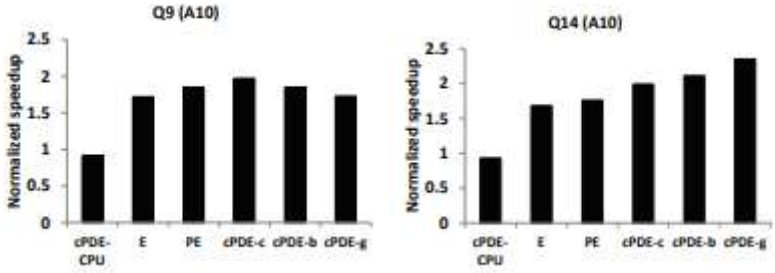
(c) Q9 on A10 (d) Q14 on A10

شکل 10



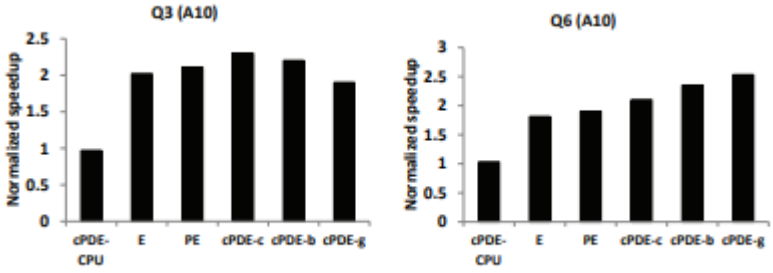

(a) Q9 results on A8 (b) Q14 results on A8

شکل 11



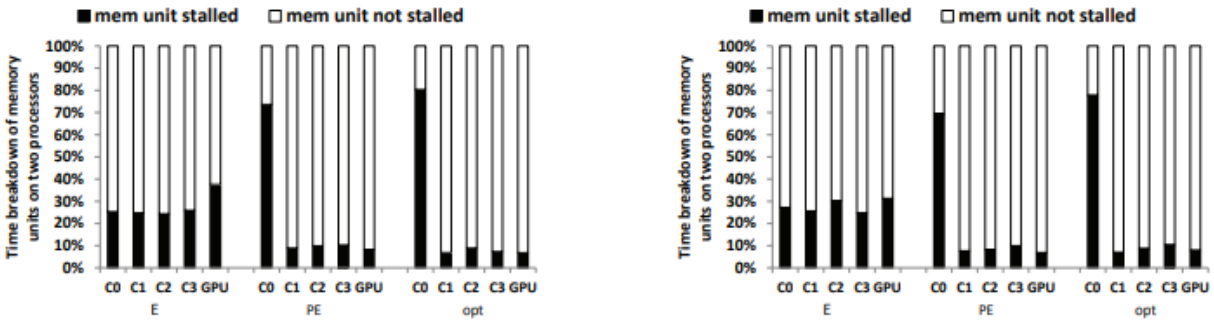

(a) Q9 results on A10 (b) Q14 results on A10

شکل 12



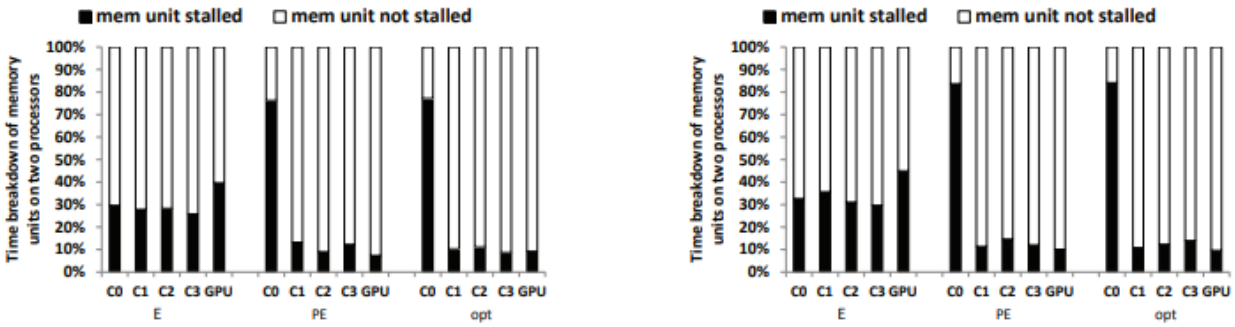

(a) Q3 results on A10 (b) Q6 results on A10

شکل 13




(a) Stall distribution of Q9 on A8 (b) Stall distribution of Q14 on A8

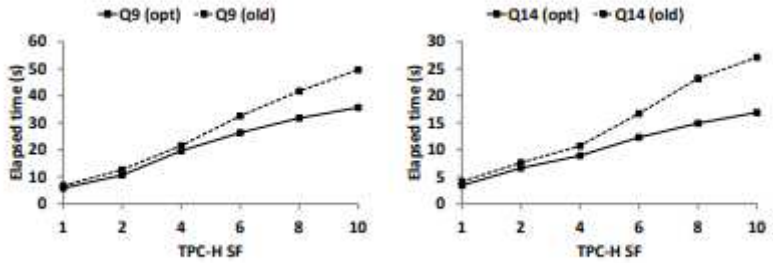
شکل 14




(a) Stall distribution of Q9 on A10

(b) Stall distribution of Q14 on A10

شکل 15




(a) Q9

(b) Q14

شکل 16

## 7. نتایج

در این مقاله، یک پارادایم پردازش مشترک پرس و جو در کش برروی معماریهای CPU-GPU جفت شده پیشنهاد کرده و از پیش واکشی به کمک CPU برای به حداقل رساندن از دست رفتن اطلاعات کش پردازش مشترک پرس و جویGPU و از طرح های غیر فشرده سازی به کمک CPU برای بهبود عملکرد اجرای پرس و جو استفاده کرده ایم. به علاوه، یک مدل هزینه برای پیش بینی زمان اجرا، پیشنهاد کرده و برنامه تخصیص هسته بهینه را انتخاب می کنیم. طبق نتایج آزمایش، پارادایم پردازش مشترک در کش، می تواند تاثیر توقف های حافظه را به شکلی موثر کاهش، و عملکرد کلی پرس و جوهای TPC-H را برروی روش دانه ریز پیشرفته AMD A8 به ترتیب 36 و 40 درصد بهبود دهد. چنین بهبودهایی نشان می دهد که پردازش مشترک پرس و جو در کش، برروی معماریهای CPU-GPU جفت شده، امیدوارکننده است.