به نام خدا

دانشکده مهندسی کامپیوتر

درس معماری کامپیوتر
نیم‌سال دوم ۰۳-۰۴
استاد: دکتر اسدی

پروژه

- پروژه در گروه‌های **چهار نفری** انجام می‌شود. نحوه گروه‌بندی در CW اطلاع‌رسانی می‌شود.
- همه موارد قابل تحویل برای پروژه را در یک فایل Zip با نام CA-Project-STDID1-STDID2-STDID3-STDID4.zip جمع‌آوری نموده و در سامانه CW بارگذاری نمایید (از هر گروه تنها یک نفر پروژه را بارگذاری نماید).
- هر کدام از پروژه‌ها را تنها ۳ گروه می‌توانند انتخاب کنند و اولویت با گروه‌هایی خواهد بود که پروژه مورد نظر را زودتر انتخاب نماید.
- در صورت هرگونه سوال یا اشکال، آن را در تالار مربوط به پروژه موردنظر در صفحه درس در CW مطرح نمایید.
- توصیه می‌شود شروع پروژه را به روزهای آخر موکول نفرمایید و در اسرع وقت کارهای اولیه پروژه را شروع نمایید.
- در صورت مشاهده تقلب کل نمرات تمرینات و پروژه صفر خواهد شد.
- استفاده از ابزارهایی مانند ChatGPT به منظور ابزار کمک آموزشی مجاز است به شرط آن که به خروجی آن اکتفا نشود.
- هر گروه باید حداکثر تا تاریخ **۲ خرداد** تیم پروژه و توصیف مختصر پروژه را در قالب یک فایل pdf تک صفحه‌ای در صفحه درس بارگذاری نماید.
- موعد انجام پروژه روز **۴ تیر** خواهد بود.
- پروژه‌ها به صورت مجازی به دستیاران آموزشی تحویل داده می‌شود. همه اعضای گروه باید برای این منظور حضور یافته و به همه‌ی قسمت‌های پروژه تسلط داشته باشند.
- گزارش پروژه باید در فرمت لاتک و در سامانه لاتک آنلاین دانشگاه نوشته شود. لذا یکی از نفرات پروژه باید قالب گزارش سمینار را از لینک ذیل انتخاب کرده و در این قالب گزارش تیم پروژه را ایجاد نماید. پروژه باید بین اعضای گروه به اشتراک گذاشته شود و تمامی اعضای گروه باید در نوشتار مشارکت نمایند. دقت شود تاریخچه مشارکت اعضای گروه، توسط دستیار آموزشی در سامانه قابل رویت خواهد بود.
  آدرس ورود به سامانه لاتک دانشگاه: Login Latex
  قالب گزارش پروژه: Template Latex
- گزارش پروژه‌های ۱ و ۵ را باید با آقایان صداقت‌گو و بهرامیان در سامانه به اشتراک گذاشته شود.
- گزارش پروژه‌های ۲ و ۴ باید با آقایان صداقت‌گو و قاسمی در سامانه به اشتراک گذاشته شود.
- گزارش پروژه‌های ۳ و ۸ باید با آقایان صداقت‌گو و علیزاده در سامانه به اشتراک گذاشته شود.

- گزارش پروژه‌های ۶ و ۷ باید با آقایان صداقت‌گو و غفوری در سامانه به اشتراک گذاشته شود.
- گزارش پروژه‌های ۹ و ۱۰ باید با آقای صداقت‌گو در سامانه به اشتراک گذاشته شود.
- ایمیل دستیاران آموزشی برای اشتراک گذاری پروژه‌ها:

صداقت‌گو: ali.sedaghatgoo43@sharif.edu
بهرامیان: bahram.mahdi83@sharif.edu
قاسمی: mohsen.ghasemi@sharif.edu
علیزاده: mohammad.alizadeh138@sharif.edu
غفوری: pouria.ghafouri83@sharif.edu

پروژه اول: طراحی واحد ممیز شناور (FPU)

مقدمه

واحد ممیز شناور (Floating-Point Unit یا FPU) یکی از اجزای حیاتی در پردازنده‌های مدرن محسوب می‌شود که امکان انجام عملیات محاسباتی بر روی اعداد اعشاری را با دقت و کارایی بالا فراهم می‌آورد. استاندارد IEEE 754 به عنوان چارچوبی جهانی برای نمایش و انجام محاسبات بر روی اعداد ممیز شناور پذیرفته شده است. در این پروژه، هدف طراحی و پیاده‌سازی یک واحد ممیز شناور ۳۲ بیتی مطابق با این استاندارد است که به پردازنده موجود اضافه خواهد شد.

صورت پروژه

در این پروژه هدف آن است که به پردازنده‌ای که پیش‌تر در تمرین‌های عملی طراحی کرده‌اید، یک واحد محاسبه اعداد ممیز شناور (FPU) اضافه شود. این واحد جدید باید توانایی انجام عملیات ریاضی روی اعداد اعشاری ۳۲ بیتی را داشته باشد، بدون آنکه عملکرد دستورات قبلی پردازنده دچار اختلال شود.
برای این منظور، لازم است مجموعه‌ای از دستورات جدید با فرمت R-Type به پردازنده افزوده شوند. کدگذاری این دستورات با استفاده از مقادیر دلخواه شما برای فیلدهای opcode و func انجام خواهد شد. فهرست این دستورات به شرح زیر است:

Instruction	Operation
FADD	rd $\leftarrow$ rs + rt
FSUB	rd $\leftarrow$ rs - rt
FMULT	rd $\leftarrow$ rs $\times$ rt
FABS	rd $\leftarrow$ |rs|
FSLT	rd $\leftarrow$ 1 if rs $<$ rt else 0

توجه کنید که این دستورات روی ثبات‌های پردازنده اصلی شما اجرا نمی‌شوند. بنابراین لازم است که یک بانک ثبات جدید مخصوص اعداد ممیز شناور ۳۲ بیتی ایجاد کنید. در نتیجه این مسئله لازم است که دو دستور با فرمت دلخواه برای خواندن و ذخیره کردن ثبات‌های FPU به پردازنده خود اضافه کنید.

گام‌های پروژه

این پروژه را در چند قسمت پیاده‌سازی کنید:

- یک بانک ثبات جدید شامل ۳۲ ثبات ۳۲ بیتی مخصوص واحد ممیز شناور ایجاد کنید. می‌توانید از ساختار بانک ثبات موجود در طراحی قبلی استفاده کرده و آن را برای این منظور گسترش دهید.
- یک واحد محاسباتی جدید مشابه ALU به پردازنده اضافه کنید که قادر به انجام عملیات ریاضی ممیز شناور باشد. این واحد باید از دستورات تعریف‌شده در بخش قبل (FADD، FSUB، FMULT، FABS، FSLT) پشتیبانی کند.
- دو دستور جدید با عملکرد زیر به پردازنده اضافه کنید:
  - TOCOP: مقدار یک ثبات از بانک ثبات اصلی را به یک ثبات در بانک ثبات ممیز شناور کپی می‌کند.
  - FROMCOP: برعکس دستور بالا، مقدار یک ثبات از بانک ثبات ممیز شناور را به یکی از ثبات‌های اصلی انتقال می‌دهد.
- سیگنال‌های کنترلی پردازنده را به گونه‌ای طراحی کنید که هنگام تشخیص دستورات ممیز شناور، واحدهای مناسب فعال شوند. ساختار کلی پردازنده باید مشابه طراحی تمرین پنجم باشد؛ به این معنا که تمام دستورات (به جز مواردی که زمان بیشتری نیاز دارند) در یک چرخه clock اجرا شوند.

- دقت کنید که پیاده‌سازی شما باید دقیقا مطابق استاندارد IEEE754 باشد و از اعداد خاص نیز مانند Inf و NaN پشتیبانی کند.
- دقت کنید که پیاده‌سازی شما در نرم‌افزار Logisim باید توانایی سنتز شدن کد Verilog برای ارزیابی داشته باشد.

گزارش و ارزیابی

- گام‌های پروژه را همراه با چالش‌های هنگام پیاده‌سازی و توضیحات هر بخش در گزارش خود توضیح دهید.
- عملکرد درست مدار را برای هر دستور به ازای چند ورودی مختلف نشان دهید. این موارد را به واسطه ایجاد یک Verilog testbench آزمایش کنید.

بخش امتیازی

یک دستور برای محاسبه سینوس از روی بسط تیلور سینوس، با تعداد جمله مشخص شده به پردازنده اضافه کنید و نحوه عملکرد و میزان خطای آن را نشان دهید.

Instruction	**Operation**
FSIN	rd $\leftarrow$ sin(rs) based on taylor series for rt steps

پروژه دوم: اضافه کردن واحد VPU به پردازنده

مقدمه

واحدهای پردازش برداری (Vector Processing Units) یکی از اجزای کلیدی در معماری‌های مدرن پردازنده‌ها به‌شمار می‌آیند که با هدف افزایش کارایی در انجام محاسبات تکراری و داده‌محور طراحی شده‌اند. برخلاف پردازش اسکالر که عملیات روی یک داده در هر سیکل انجام می‌شود، در پردازش برداری مجموعه‌ای از داده‌ها به‌صورت همزمان پردازش می‌شوند. این ویژگی باعث کاهش چشمگیر در تعداد سیکل‌های لازم برای اجرای حلقه‌های محاسباتی، بهبود بهره‌وری پردازنده و افزایش توان عملیاتی در کاربردهایی مانند پردازش تصویر، رمزنگاری، یادگیری ماشین و شبیه‌سازی‌های علمی می‌شود.

مزیت اصلی VPUها در استفاده بهینه از منابع سخت‌افزاری و کاهش سربار ناشی از تکرار دستورالعمل‌های مشابه است. با بهره‌گیری از عملیات موازی روی عناصر بردار، نه تنها سرعت اجرای برنامه‌ها افزایش می‌یابد، بلکه مصرف انرژی نیز نسبت به اجرای ترتیبی کاهش می‌یابد، که این امر به‌ویژه در طراحی پردازنده‌های تعبیه‌شده و سیستم‌های کم‌مصرف اهمیت ویژه‌ای دارد.

در این پروژه، با هدف آشنایی عملی با مفاهیم پردازش برداری، قصد داریم به پردازنده‌ی طراحی‌شده در تمرین‌های قبلی، یک واحد پردازش برداری برای اعداد صحیح اضافه کنیم. این واحد قابلیت انجام عملیات جمع، تفریق و عملگرهای منطقی روی بردارهای 128 بیتی از اعداد 16 و 32 بیتی را خواهد داشت و شامل مجموعه‌ای از ثبات‌ها و دستورالعمل‌های جدید خواهد بود که به معماری موجود افزوده می‌شوند.

صورت پروژه

در این پروژه قصد داریم که به پردازنده‌ای که در تمرین‌های عملی طراحی کرده‌اید، یک واحد پردازش برداری اعداد صحیح (VPU) اضافه کنیم. این واحد جدید باید توانایی انجام عملیات ریاضی روی بردارهای 128 بیتی از اعداد صحیح 32 و 16 بیتی را داشته باشد، بدون آنکه عملکرد دستورات قبلی پردازنده دچار اختلال شود.

برای این منظور، لازم است مجموعه‌ای از ثبات‌ها و دستورات جدید با فرمت R-Type به پردازنده افزوده شوند. کدگذاری این دستورات با استفاده از مقادیر دلخواه شما برای فیلدهای opcode و func انجام خواهد شد. فهرست این دستورات به شرح زیر است:

Mnemonic	Operation
V32.MUL	rd[i] ← rs[i] × rt[i]
V32.ADD	rd[i] ← rs[i] + rt[i]
V32.SUB	rd[i] ← rs[i] - rt[i]
V16.MUL	rd[i] ← rs[i] × rt[i]
V16.ADD	rd[i] ← rs[i] + rt[i]
V16.SUB	rd[i] ← rs[i] - rt[i]
VAND	rd[i] ← rs[i] & rt[i]
VOR	rd[i] ← rs[i] | rt[i]
VXOR	rd[i] ← rs[i] ^ rt[i]
VNOT	rd[i] ← ~rs[i]

توجه کنید که این دستورات روی ثبات‌های پردازنده اصلی شما اجرا نمی‌شوند. بنابراین لازم است که یک بانک ثبات جدید مخصوص اعداد ممیز شناور 32 بیتی ایجاد کنید. در نتیجه این مسئله لازم است که دو دستور با فرمت دلخواه برای خواندن و ذخیره کردن ثبات‌های VPU به پردازنده خود اضافه کنید.

گام‌های پروژه

این پروژه را در چند قسمت پیاده‌سازی کنید:

- یک بانک ثبات جدید شامل 8 ثبات 128 بیتی مخصوص واحد پردازش برداری ایجاد کنید. می‌توانید از ساختار بانک ثبات موجود در طراحی قبلی استفاده کرده و آن را برای این منظور گسترش دهید.
- یک واحد محاسباتی جدید مشابه ALU به پردازنده اضافه کنید که قادر به انجام عملیات ریاضی موازی باشد. این واحد باید از دستورات تعریف‌شده در بخش قبل پشتیبانی کند.
- دو دستور جدید با عملکرد زیر به پردازنده اضافه کنید
  - TOCOP: مقدار یک ثبات از بانک ثبات اصلی را به یک ثبات در بانک ثبات ممیز شناور کپی می‌کند.
  - FROMCOP: برعکس دستور بالا، مقدار یک ثبات از بانک ثبات ممیز شناور را به یکی از ثبات‌های اصلی انتقال می‌دهد.
- سیگنال‌های کنترلی پردازنده را به گونه‌ای طراحی کنید که هنگام تشخیص دستورات برداری، واحدهای مناسب فعال شوند. ساختار کلی پردازنده باید مشابه طراحی تمرین پنجم باشد؛ به این معنا که تمام دستورات (به جز مواردی که زمان بیشتری نیاز دارند) در یک چرخه کلاک اجرا شوند.
- دقت کنید که پیاده‌سازی شما در نرم‌افزار Logisim باید توانایی سنتز شدن کد Verilog برای ارزیابی داشته باشد.

بخش امتیازی

دستورات اختصاصی بارگیری و بارگذاری را برای این واحد پیاده کنید که با یک دستور کل 128 بیت را بارگیری یا بارگذاری کند.

پروژه سوم: پردازنده دوهسته‌ای

مقدمه

با پیشرفت روزافزون نیازهای محاسباتی در حوزه‌های مختلف، استفاده از پردازنده‌های چند هسته‌ای (Multi-core Processors) به یک راه‌حل رایج و مؤثر برای افزایش کارایی سیستم‌های پردازشی تبدیل شده است. در این معماری، چند هسته پردازشی به صورت هم‌زمان و مستقل از یکدیگر عملیات انجام می‌دهند و در عین حال می‌توانند منابع خاصی نظیر حافظه داده را به اشتراک بگذارند. این ساختار باعث افزایش موازی‌سازی (Parallelism) در اجرای برنامه‌ها، بهبود زمان پاسخ‌گویی و بهره‌وری بالاتر سیستم می‌شود.

در این پروژه، هدف ایجاد یک سیستم دو هسته‌ای ساده است که هر هسته آن مشابه پردازنده چندهسته‌ای طراحی شده در تمارین عملی است. این دو هسته به صورت کاملاً مستقل عمل می‌کنند و تنها در حافظه داده با یکدیگر اشتراک دارند. بررسی نحوه تعامل این دو هسته با حافظه مشترک، مدیریت هم‌زمانی و تحلیل عملکرد چنین سیستمی، محور اصلی این پروژه خواهد بود.

اهداف

۱. آشنایی با معماری پردازنده‌های چند هسته‌ای Multi-core Processors و مزایای آن‌ها در افزایش کارایی سیستم

۲. طراحی و پیاده‌سازی یک سیستم ساده دو هسته‌ای با استفاده از معماری مشابه تمرین‌های عملی

۳. بررسی شیوه اشتراک‌گذاری حافظه داده بین هسته‌ها و مدیریت هم‌زمانی[1]

صورت پروژه

در این پروژه هدف طراحی و پیاده‌سازی یک سیستم ساده دوهسته‌ای با الهام از معماری پردازنده مالتی‌سایکل طراحی‌شده در طول ترم است. در این سیستم، هر هسته شامل اجزای مستقلی مانند PC، ALU، فایل ثبات و واحد کنترل است، درحالی‌که حافظه داده به صورت مشترک بین آن‌ها تعریف می‌شود. یکی از چالش‌های اصلی، مدیریت صحیح دسترسی هم‌زمان به این حافظه است که با استفاده از اولویت‌بندی یا طراحی بافر واسط برای عملیات نوشتن، حل خواهد شد. همچنین، دستورهای جدیدی مانند cpuid برای شناسایی هسته اجرایی و sync برای همگام‌سازی اجرای دو پردازنده پیاده‌سازی می‌شوند.

گام های پروژه

گام اول: ایجاد ساختار پایه معماری جدید با ویژگی‌های زیر:

- کپی کردن طراحی پردازنده مالتی‌سایکل قبلی برای ساخت دو هسته مستقل (مثلاً CPU0 و CPU1).
- هر پردازنده Register File، PC، ALU و کنترلر خودش را دارد.
- تنها حافظه داده به صورت مشترک بین دو پردازنده تعریف می‌شود.
- تنظیم سیگنال‌های کنترلی حافظه به‌گونه‌ای که در صورت نیاز همزمان دو پردازنده به حافظه داده، پردازنده اول (CPU0) همیشه اولویت نوشتن و خواندن داشته باشد.

(البته می‌توانید خواندن از حافظه را به این صورت مدیریت کنید که یکی از پردازنده‌ها همواره در سطح ۰ کلاک و دیگری در سطح ۱ کلاک داده از حافظه بخواند. نحوه پیاده‌سازی وابسته به تصمیم تیم اخذکننده پروژه است.)

گام دوم: طراحی بافر واسط برای نوشتن در حافظه:

- ساخت یک بافر FIFO بین پردازنده‌ها و حافظه برای عملیات نوشتن[2].

[1] Concurrency
[2] Write buffer

- پردازنده‌ها داده و آدرس را در بافر قرار می‌دهند و حافظه داده به نوبت و کنترل‌شده اطلاعات را از بافر دریافت می‌کند.
- توجه کنید که درصورتی‌که هر دو پردازنده می‌خواهند به حافظه بنویسند، باید دو عنصر به بافر نوشتن اضافه شود.
- این طراحی از تداخل نوشتن هم‌زمان جلوگیری می‌کند و به هماهنگی حافظه کمک می‌کند.

گام سوم: پیاده‌سازی دستور cpuid rd:

- افزودن دستور جدید cpuid به دیکودر و واحد کنترل.
- این دستور باید شناسه یکتای هر پردازنده (مثلاً ۰ برای CPU0 و ۱ برای CPU1) را در یک ثبات که به‌عنوان ورودی می‌گیرد، قرار دهد.
- توجه کنید که باید sp هرکدام از پردازنده‌ها به جای متفاوتی از حافظه اشاره کند، وگرنه ممکن است موجب تداخل شود. برای پیاده‌سازی این مورد می‌توانید با چک کردن خروجی cpuid مقدار sp را تعیین کنید.

گام چهارم: پیاده‌سازی دستور sync:

- تعریف دستور جدید sync در دیکودر و کنترل.
- اجرای دستور به صورت مسدودکننده (blocking):

  وقتی یک پردازنده به sync می‌رسد، منتظر می‌ماند تا پردازنده دیگر هم به این دستور برسد. زمانی که هر دو هسته به sync رسیدند، ادامه اجرای هر دو آزاد می‌شود.[۳]

قسمت امتیازی

- نشان دهید که در صورت دسترسی هم‌زمان دو پردازنده به بخش مشترکی از حافظه، ممکن است محاسبات به‌درستی انجام نشوند.
- دستور زیر را برای پردازنده خود پیاده‌سازی کنید (تخصیص opcode بر عهده شماست):

  exchng rt, [rs + imm] : به‌صورت هم‌زمان و یک‌باره، مقدار ثبات rt و خانه حافظه rs + imm را تعویض می‌کند.
- سپس با استفاده از این دستور، یک spinlock برای همگام‌سازی میان دو هسته طراحی کنید.
- توجه داشته باشید که پیاده‌سازی این دستور باید به‌گونه‌ای باشد که در هنگام اجرای آن توسط یک پردازنده، پردازنده دیگر قادر به نوشتن در حافظه نباشد.
- توضیح خلاصه spin_lock:

```
lock_mem

lock(atomic_val)
  reg = 1
  while(exchng(reg, lock_mem) == 1) nop;
release(atomic_val)
  lock_mem = 0
```

[۳] این دستور برگرفته از دستور مشابهی در معماری پردازنده‌های گرافیکی است.

نحوه‌ی تحویل گزارش

۱. گام‌های پروژه را همراه با چالش‌های هنگام پیاده‌سازی و توضیحات هر بخش در گزارش خود توضیح دهید.

۲. برای بررسی تأثیر استفاده از معماری دوهسته‌ای در مقایسه با پردازنده تک‌هسته‌ای طراحی‌شده در طول ترم، لازم است یک برنامه مشخص به‌صورت مشترک در هر دو سیستم پیاده‌سازی شود. این برنامه شامل ضرب دو ماتریس ۸ در ۸ و محاسبه مجموع عناصر حاصل‌ضرب آن‌هاست. ابتدا دو ماتریس از حافظه خوانده می‌شوند و حاصل‌ضرب آن‌ها در یک ماتریس دیگر نوشته می‌شود. سپس مجموع تمامی عناصر این ماتریس حساب شده و در خانه‌ای مشخص از حافظه ذخیره می‌شود.

در نسخه تک‌هسته‌ای، تمام مراحل توسط یک پردازنده انجام خواهد شد. اما در نسخه دوهسته‌ای، امکان تقسیم کار بین دو پردازنده وجود دارد؛ مثلاً هر پردازنده مسئول محاسبه چهار سطر از ماتریس ضرب باشد. عملیات جمع نهایی نیز می‌تواند یا توسط یکی از پردازنده‌ها انجام شود یا ابتدا به صورت موازی انجام گرفته و سپس با استفاده از دستور sync هماهنگ شود.

در پایان، تعداد سیکل‌های کلاک صرف‌شده برای اجرای کامل برنامه در هر دو نسخه اندازه‌گیری شده و مقایسه می‌شود تا تاثیر پردازش موازی در عملکرد کلی سیستم مشخص گردد.

۳. برای بخش امتیازی، ضمن نشان دادن اشتباه محاسباتی به واسطه دسترسی همزمان در یک نمونه کد، از قفل ساخته شده استفاده کرده و نسخه‌ای که در آن اشتباه محاسباتی ایجاد نمی‌شود را پیاده‌سازی و ارائه کنید.

پروژه چهارم: طراحی حافظه نهان برای پردازنده

مقدمه

در معماری کامپیوترهای مدرن، حافظه‌های نهان (Cache) نقش حیاتی در کاهش شکاف سرعت بین پردازنده و حافظه اصلی ایفا می‌کنند. این پروژه به طراحی و پیاده‌سازی یک سلسله‌مراتب حافظه نهان برای پردازنده توسعه داده شده در طول ترم می‌پردازد. هدف اصلی بهبود عملکرد پردازنده از طریق کاهش تاخیر دسترسی به حافظه و بهینه‌سازی استفاده از پهنای باند حافظه است.

صورت پروژه

در این پروژه قصد داریم به پردازنده‌ای که در طول ترم توسعه داده‌اید، یک حافظه نهان اضافه کنیم. هدف نهایی ما از این پروژه، تسریع برنامه‌ها در پردازنده است به طوری که این تسریع به صورت شفاف قابل بررسی باشد. حافظه‌ای که در طول ترم از آن استفاده می‌کردید، خواندن و نوشتن را در یک چرخه انجام می‌داد اما در عمل این اتفاق رخ نمی‌دهد و بیش از یک چرخه برای انجام عملیات حافظه نیاز است.

گام‌های پروژه

این پروژه را در چند قسمت پیاده‌سازی کنید:

- ابتدا لازم حافظه داده خود را به گونه‌ای تغییر دهید که هر عملیات رو در **چهار** چرخه انجام دهد. می‌توانید از قرار دادن چند ثبات در ورودی حافظه و اتصال سری آنها به یکدیگر استفاده کنید (در این پروژه فرض شده است که حافظه دستورات همچنان در یک چرخه عملیات خواندن و نوشتن را انجام می‌دهد).
- سپس لازم است که پردازنده خود را به گونه‌ای تغییر دهید که هنگام انجام دستورات حافظه، پردازنده متوقف شود و تا پایان عملیات حافظه دستور جدیدی انجام نشود (تغییرات شما باید روی پردازنده تمرین ۵ با قابلیت پشتیبانی از خط لوله انجام شود).
- یک حافظه نهان سطح ۱ به پردازنده خود اضافه کنید تا بر آن اساس دسترسی‌های به حافظه را به حداقل برسانید. این حافظه نهان باید ویژگی‌های زیر را داشته باشد:
  - نحوه نوشتن از نوع write back باشد.
  - برای هر بلوک یک dirty bit نگه‌داری شود و در صورتی که مقدار آن ۱ نبود، write back انجام نشود (تاثیر این موضوع در حالتی که یک جایگزینی رخ دهد باید مشهود باشد. به طوری که اگر این بیت ۰ بود، عملیات خواندن همراه با جایگزینی ۴ چرخه و اگر این بیت ۱ بود، عملیات خواندن همراه با جایگزینی ۸ چرخه طول می‌کشد. پیاده سازی این بخش اختیاری است و نمره‌ای، چه اجباری و چه امتیازی ندارد).
  - مدل نگاشت حافظه نهان به صورت نگاشت مستقیم باشد.
  - تمام پیاده‌سازی شما در نرم‌افزار لاجیسیم باید قابلیت سنتز شدن به کد وریلاگ داشته باشند.

گزارش و ارزیابی

- در گزارش خود نحوه پیاده‌سازی و چالش‌های هر مرحله را توضیح دهید.
- برای هر کدام از سناریوهای ممکن در استفاده از حافظه نهان، یک برنامه کوچک طراحی کنید و رفتار انتظاری را با عملکرد پیاده‌سازی خود مقایسه کنید.
- دو برنامه مشابه قطعه کدهای زیر را بر روی پردازنده خود اجرا کنید و با شمارش کلاک‌های مورد نیاز برای اتمام هر کدام از برنامه‌ها تاثیر استفاده نادرست از حافظه نهان را به عنوان یک برنامه‌نویس در قالب یک نمودار نشان دهید.

```
#define N 16 // Should be large enough to show the difference

int a[N][N];
int sum = 0;

// P1
for (int i; i < N; i++)
  for(int j; j < N; j++)
    sum += a[i][j];

// P2
for (int i; i < N; i++)
  for(int j; j < N; j++)
    sum += a[j][i];
```

بخش امتیازی

در این بخش باید یک لایه به حافظه نهان خود اضافه کنید و یک حافظه نهان سطح در پردازنده خود به طوری که اندازه آن چهار برابر حافظه سطح ۱ باشد، داشته باشید. فرض کنید هر عملیات در این حافظه دو چرخه طول می‌کشد. این حافظه مانند حافظه قبل از مدل نگاشت مستقیم استفاده می‌کند. اما نحوه نوشتن آن بر روی حافظه به صورت write through است. در نهایت یک برنامه در پردازنده خود اجرا کنید و با شمارش چرخه‌های آن، عملکرد پردازنده را در مقایسه با حافظه نهان تک لایه مقایسه کنید.

پروژه پنجم: پیاده‌سازی پروتکل I/O برای پردازنده

مقدمه

امروزه، پردازنده‌ها برای تعامل با دنیای خارج نیاز به رابط‌های ورودی/خروجی[۴] دارند. پروتکل‌های سریال مانند UART و I2C به‌عنوان استانداردهای صنعتی، امکان ارتباط کارآمد بین پردازنده و دستگاه‌های جانبی (مانند سنسورها، نمایشگرها و حافظه‌ها) را فراهم می‌کنند. در این پروژه، هدف طراحی و پیاده‌سازی یک ماژول I/O برای پردازنده MIPS است که از یکی از این پروتکل‌ها پشتیبانی کند. این ماژول باید قابلیت اتصال به گذرگاه پردازنده را داشته باشد و امکان تبادل داده با محیط خارج را به‌صورت پایدار و قابل برنامه‌ریزی ارائه دهد.

صورت پروژه

در این پروژه قصد داریم به پردازنده خود، درگاه ورودی و خروجی اضافه کنیم که بتوانیم با ماژول های جانبی به طور موثر ارتباط برقرار کنیم.

گام‌های پروژه

گام اول: انتخاب پروتکل ورودی/خروجی: برای این درگاه شما می‌بایست یک پروتکل صنعتی سریال (مانند UART و I2C) را انتخاب کنید. توجه کنید که نحوه کارکرد این پروتکل را باید در گزارش خود توضیح دهید.

گام دوم: پیاده‌سازی ماژول فرستنده و گیرنده برای پروتکل انتخاب شده.

گام سوم: متصل کردن ماژول‌های فرستنده و گیرنده به پردازنده.

گام چهارم: پیاده‌سازی روشی برای پردازنده که با این درگاه‌های فرستنده و گیرنده ارتباط گیرد. برای این‌کار می‌توانید از روش‌های «نگاشت به حافظه» و یا «دستورات اختصاصی» استفاده کنید.

گام پنجم: برنامه‌ای بنویسید که با استفاده از ۴ LED، به ترتیب توالی فیبوناچی را نمایش دهد. همین‌طور یک دکمه در نظر بگیرید که با استفاده از آن به ابتدای توالی بازگردد.

قسمت امتیازی

از پروتکل پیاده‌شده برای ارتباط بین پردازنده و یک ماژول صفحه کلید، تعدادی LED و TTY استفاده کنید. سپس برنامه‌ای بنویسید که بتواند دستورات مورد نظر کاربر را از طریق صفحه کلید گرفته و آن‌ها را روی صفحه‌نمایش TTY نمایش دهد و در نهایت نتیجه اجرای این دستورات را با استفاده از LEDها و یا خروجی TTY نشان دهد. توصیف برنامه و دستورات به عهده خود شماست. البته برنامه قسمت قبل مبتنی بر حساب کردن فیبوناچی نیز در اینجا به عنوان یک دستور باید قابل اجرا باشد.

یک پیاده‌سازی نمونه برای دستورات

- **FIB:** برنامه قسمت قبل را اجرا کن، همین‌طور برای خاتمه برنامه نیز یک کلید مشخص تعیین کنید.
- **LED x ON:** چراغ x را روشن کن.
- **LED x OFF:** چراغ x را خاموش کن.

[۴] I/O

- **TEXT {text}:** عبارت {text} را در خط بعدی TTY نمایش بده.

توجه کنید که می‌توانید فرض کنید ورودی نادرست نداریم و همین‌طور نام و ساختار دستورات به عهده خودتان است.

پروژه ششم: پیاده‌سازی سیاست‌های جایگزینی حافظه نهان در شبیه‌ساز ChampSim

مقدمه

با افزایش سرعت پردازنده‌ها و محدودیت‌های سرعت حافظه‌های اصلی، شکاف عملکردی بین این دو به چالشی بزرگ در طراحی سیستم‌های کامپیوتری تبدیل شده است. حافظه‌های نهان به عنوان لایه‌ای میانی، در کاهش تأخیر دسترسی به حافظه اصلی و بهبود عملکرد سیستم نقش کلیدی ایفا می‌کنند.
سیاست‌های جایگزینی حافظه نهان تصمیم می‌گیرند که کدام بلوک داده حذف و جایگزین شود. انتخاب بهینه این الگوریتم‌ها تأثیر زیادی بر کارایی سیستم دارد. در این پروژه، هدف پیاده‌سازی و مقایسه چند سیاست جایگزینی حافظه نهان است تا عملکرد آنها در سناریوهای مختلف تحلیل شود و به درک بهتری از مدیریت حافظه نهان در سیستم‌های کامپیوتری برسیم.

اهداف

۱. آشنایی با شبیه‌ساز ChampSim و نحوه کار با آن

۲. آشنایی با سیاست‌های جایگزینی مختلف در حافظه نهان

۳. مقایسه و تحلیل عملکرد سیاست‌های مختلف حافظه نهان بر عملکرد کلی سامانه

صورت پروژه

در این پروژه قصد داریم تا چهار سیاست جایگزینی حافظه نهان را در شبیه‌سازی ChampSim پیاده‌سازی کنیم و با استفاده از برنامه‌های محک مختلف، عملکرد آن‌ها را با یکدیگر مقایسه کنیم.

- LFU (Least Frequency Used)
- Tree-PLRU
- ARC (Adaptive replacement cache)
- MRU (Most Recently Used)

گام‌های پروژه

گام اول: لینک‌های مرتبط با هر سیاست را مطالعه کنید تا با نحوه کارکرد و پیاده‌سازی آن‌ها آشنا شوید.

گام دوم: درک خود از نحوه چگونگی عملکرد این سیاست‌ها را در گزارش ارائه دهید.

گام سوم: این سیاست‌ها را در شبیه‌ساز ChampSim پیاده‌سازی کنید. حتما نحوه پیاده‌سازی باقی سیاست‌های جایگزینی که در این شبیه‌ساز استفاده شده‌اند را مشاهده کنید تا با ساختار اولیه نحوه نوشتن کد در این شبیه‌ساز آشنا شوید.

گام چهارم: شبیه‌ساز را با استفاده از این سیاست‌ها و سیاست‌های LRU، Random و SRRIP که توسط خود شبیه‌ساز پیاده شده است، پیکربندی کرده و برنامه‌های محک مختلف را بر روی آن اجرا کنید.

گام پنجم: عملکرد این سیاست‌ها را با پارامترهای مختلفی که شبیه‌ساز به شما می‌دهد با یکدیگر مقایسه کنید.

قسمت امتیازی

برای بخش امتیازی باید سیاست LARC[5] را پیاده‌سازی کنید. این سیاست همان سیاست ARC است با این تفاوت که به محض اتفاق افتادن Cache Miss حافظه نهان را به‌روزرسانی نمی‌کند.

نحوه‌ی تحویل گزارش

تمامی موارد ذیل باید در یک پوشه ذخیره و تحویل داده شوند:

۱. کدهای توسعه داده‌شده و توضیحات لازم جهت اجرای مجدد آن‌ها

۲. گزارش کامل از کارهای انجام شده و توضیحات کدهای نوشته

۳. نتایج مربوط به بررسی عملکرد هرکدام از سیاست‌ها

۴. تحلیل و مقایسه نتایج سیاست‌های مختلف

۵. همچنین در گزارش خود به طور خلاصه درمورد پیچیدگی پیاده‌سازی این سیاست‌ها به صورت سخت‌افزاری و هزینه پیاده‌سازی آن‌ها بحث کنید.

[5] Lazy Adaptive Replacement Cache

پروژه هفتم: پیاده‌سازی TAGE Branch Predictor در شبیه‌ساز ChampSim

مقدمه

در پردازنده‌های مدرن، یکی از چالش‌های اصلی برای افزایش کارایی و بهبود بهره‌وری، پیش‌بینی درست مسیر اجرای شاخه‌ها[6] در کد برنامه است. شاخه‌ها نقاطی هستند که جریان اجرای برنامه می‌تواند بسته به شرط‌های منطقی به مسیرهای مختلفی هدایت شود. پیش‌بینی شاخه به پردازنده کمک می‌کند تا زودتر تصمیم بگیرد کدام مسیر باید اجرا شود و بدین ترتیب از توقف یا تاخیر در اجرای دستورالعمل‌ها جلوگیری کند.
در این پروژه، هدف اصلی طراحی و پیاده‌سازی مدل TAGE[7] به عنوان یک Branch Predictor است تا با استفاده از الگوریتم‌های پیشرفته، بتوان پیش‌بینی‌های دقیق‌تر و بهینه‌تری انجام داد و تاثیر آن را بر عملکرد پردازنده ارزیابی کرد.

اهداف

۱. آشنایی با Branch Predictorها و تاثیر آن‌ها در عملکرد پردازنده

۲. مقایسه و تحلیل عملکرد Branch Predictorهای مختلف

۳. پیاده‌سازی یک Branch Predictor برای آشنایی با ساختار و نحوه عملکرد آن‌ها

صورت پروژه

در این پروژه قصد داریم تا TAGE Branch Predictor را در شبیه‌ساز ChampSim پیاده‌سازی کنیم. برای آشنایی بیشتر با TAGE از این لینک استفاده کنید.
به طور خلاصه TAGE یک Branch Predictor پویا است که در هنگام اجرای برنامه داده‌های خود را به‌روز کرده و بر اساس آن‌ها برای یک شاخه تصمیم می‌گیرد. این سیاست شامل یک پیش‌بینی‌کننده پایه (مانند bimodal) است و دارای چندین جدول با عمق متفاوت است که داده‌های مربوط به پیش‌بینی را داخل خود ذخیره می‌کند.
این سیاست به ازای هر شاخه یک tag را با استفاده از یک Hash Function محاسبه کرده و از جدول با عمق بیشتر شروع کرده و هرگاه یک tag مشابه در یک جدول پیدا کرد، جواب آن را برمی‌گرداند و در صورتی که آن tag در جدول پیدا نشد، سراغ جدول بعدی می‌رود و در صورتی که جداول تمام شوند، نتیجه را از پیش‌بینی‌کننده پایه برمی‌گرداند.
این جداول شامل بیت‌های tag، u و ctr هستند. بیت tag برای پیدا کردن یک entry مناسب استفاده می‌شود. ctr برای پیش‌بینی taken و یا not-taken بودن استفاده می‌شود و u نشان دهنده کاربردی بودن آن entry از جدول است و برای جایگزین کردن مقادیر داخل جدول استفاده می‌شود.

گام‌های پروژه

گام اول: مطالعه مقاله و آشنایی با این Branch Predictor و نحوه عملکرد آن

گام دوم: بررسی نحوه پیاده‌سازی Branch Predictor در این شبیه‌ساز برای آشنایی با ساختار اولیه نوشتن کد

گام سوم: پیاده‌سازی TAGE در این شبیه‌ساز. برای Base Predictor می‌توانید از سیاست Bimodal در ChampSim استفاده کنید.

گام چهارم: بررسی عملکرد این Branch Predictor و مقایسه آن با سایر Branch Predictorها که در این شبیه‌ساز استفاده شده‌اند. (مانند Bimodal و Perceptron)

[6] Branch
[7] TAgged GEometric

قسمت امتیازی

برای بخش امتیازی باید پارامترهای بهینه را برای این Branch Predictor محاسبه کنید. برای این کار می‌توانید TAGE را با پارامترهای مختلف شبیه‌سازی کنید و خروجی‌های ChampSim را به ازای آن پارامترها با یکدیگر مقایسه کنید. پارامترهای پیشنهادی شامل تعداد جداول، عمق آن‌ها، اندازه Tag و مواردی است که به نظر شما می‌تواند در عملکرد آن تاثیر بگذارد.
دقت کنید که حتما باید چندین پارامتر بر روی برنامه‌های محک مختلف اجرا شود تا بتوانید عملکرد آن را در حالات مختلف بررسی کنید. همچنین در نهایت باید برای پارامترهای بهینه به‌دست آمده برای هر برنامه محک، Base Predictor را نیز تغییر دهید (برای این کار می‌توانید از Predictorهای داخل ChampSim استفاده کنید) و عملکرد آن‌ها را نیز با یکدیگر مقایسه کنید. دقت کنید که بخشی از نمره به شیوه مستندسازی شما وابسته است و پیشنهاد می‌شود عملکرد موارد مختلف را در قالب نمودارهایی با یکدیگر مقایسه کنید تا تحلیل‌ها به سادگی انجام شود.

نحوه‌ی تحویل گزارش

تمامی موارد ذیل باید در یک پوشه ذخیره و تحویل داده شوند:

۱. کدهای توسعه داده‌شده و توضیحات لازم جهت اجرای مجدد آن‌ها

۲. گزارش کامل از کارهای انجام‌شده و توضیحات کدهای نوشته

۳. نتایج مربوط به بررسی عملکرد این Branch Predictor

۴. تحلیل و مقایسه نتایج عملکرد Branch Predictorهای دیگر در مقایسه با TAGE

۵. برای بخش امتیازی باید نتایج عملکرد را به ازای تمامی پارامترها و Base Predictorها به ازای برنامه‌های محک مختلف، گزارش دهید.

پروژه هشتم: پردازش نزدیک حافظه

مقدمه

در معماری‌های کلاسیک کامپیوتر، پردازنده و حافظه از یکدیگر جدا هستند و دسترسی به داده‌ها نیازمند انتقال بین این دو بخش است. با پیشرفت پردازنده‌ها و افزایش سرعت آن‌ها، یک شکاف عملکردی به وجود آمده که به آن دیوار حافظه[8] گفته می‌شود؛ به این معنا که سرعت پردازنده‌ها بسیار بیشتر از سرعت حافظه (به‌ویژه حافظه‌های اصلی مثل DRAM) افزایش یافته و در نتیجه زمان زیادی صرف انتظار برای دریافت یا نوشتن داده‌ها در حافظه می‌شود.

در بسیاری از برنامه‌ها، زمانی که سیستم در حال پردازش داده نیست یا نیازی به خواندن/نوشتن به حافظه ندارد، حافظه عملاً بلااستفاده باقی می‌ماند. این موضوع در برنامه‌هایی که تعامل زیادی با حافظه ندارند بیشتر دیده می‌شود، یعنی پردازنده درگیر محاسبات داخلی است ولی حافظه بدون استفاده مانده. این بلااستفاده بودن ظرفیت حافظه یک فرصت از دست‌رفته محسوب می‌شود.

از طرف دیگر، در برنامه‌هایی که نیازمند پردازش حجم زیادی از داده هستند (مانند پردازش تصویر، یادگیری ماشین یا تحلیل‌های علمی)، بار سنگینی روی سیستم حافظه وارد می‌شود. در این موارد، حجم بالایی از دستورات load و store تولید می‌شود و باعث سربار بالا، افزایش مصرف انرژی، و به‌هم‌ریختگی کش‌ها می‌گردد.

پردازش نزدیک حافظه[9] یکی از راه‌حل‌های ارائه‌شده برای کاهش این مشکلات است. در این رویکرد، بخش‌هایی از عملیات پردازشی در نزدیکی حافظه (یا حتی در داخل چیپ‌های حافظه) انجام می‌شوند. این باعث کاهش نیاز به انتقال مکرر داده بین حافظه و پردازنده، کاهش تأخیر دسترسی به داده، و کاهش مصرف انرژی می‌شود.

اهداف

- درک چالش‌های موجود در معماری‌های سنتی پردازنده‌ـ‌محور، به‌ویژه مشکل دیوار حافظه
- آشنایی با مفهوم پردازش نزدیک حافظه و اهمیت آن در کاربردهای داده‌محور
- مطالعه ساختار حافظه‌های نوین و قابلیت‌های آن‌ها برای اجرای عملیات محاسباتی
- تحلیل نحوه‌ی پیاده‌سازی و ارزیابی یک نمونه‌ی ساده از سیستم‌های NMC

صورت پروژه

در این پروژه، هدف طراحی و پیاده‌سازی یک واحد محاسباتی نزدیک حافظه و اتصال آن به پردازنده‌ی طراحی‌شده در طول ترم است. این واحد باید قادر به اجرای عملیات‌های پایه‌ای مانند add، sub، neg و در حالت امتیازی mul روی بلوک‌های حافظه باشد. در ادامه، مجموعه‌ای از دستورالعمل‌های جدید به ISA افزوده می‌شود تا تعامل کامل با این واحد ممکن گردد و کارایی سیستم مورد ارزیابی قرار گیرد.

گام‌های پروژه

گام اول: در این گام، هدف طراحی یک واحد محاسباتی مستقل است که عملکرد آن مطابق ویژگی‌های زیر باشد:

- این واحد باید دارای سه ثبات برای نگهداری آدرس یا شناسه بلوک‌های حافظه باشد که از طریق یک ورودی مشخص و به صورت سینکرون مقداردهی می‌شوند. این ثبات‌ها به ترتیب با نام‌های src1، src2 و dst شناخته می‌شوند.

[8] Memory wall
[9] Near-Memory Computation (NMC)

- لازم است مکانیزمی برای انتخاب دستورالعمل مورد نظر جهت اجرا در این واحد در نظر گرفته شود. نحوه پیاده‌سازی این انتخاب به تصمیم تیم اخذکننده پروژه واگذار شده است، اما پیشنهاد می‌شود یک ثبات مجزا برای ذخیره نوع دستور در نظر گرفته شود (مشابه ثبات‌های ورودی قبلی).
- باید دستورالعملی کنترلی وجود داشته باشد که با مراجعه به مقادیر فعلی موجود در ثبات‌های src1، src2، dst و همچنین دستورالعمل انتخاب‌شده (بخش قبل)، عملیات مورد نظر را بر روی بلوک‌های مشخص‌شده از حافظه اجرا کند. لیست عملیات قابل انجام در ادامه قرار گرفته است.

  – **توجه:** اجرای این دستور باید فقط بر اساس مقادیری صورت گیرد که در لحظه دریافت سیگنال شروع تنظیم شده‌اند، و تغییر احتمالی این ثبات‌ها در طول اجرای عملیات نباید اثری بر فرآیند جاری داشته باشد.

- این واحد محاسباتی باید دارای یک خروجی کنترلی busy باشد که در هنگام اجرای عملیات مقدار آن برابر ۱ بوده و در سایر مواقع صفر باشد. زمانی که این خروجی برابر صفر است، واحد نباید هیچ‌گونه دسترسی‌ای به حافظه داشته باشد. با رسیدن سیگنال شروع (Start)، مقدار این خروجی باید به ۱ تغییر کند. همچنین، واحد محاسباتی باید این قابلیت را داشته باشد که با هر بار دریافت سیگنال شروع، عملیات جدیدی را با استفاده از مقادیر فعلی ثبات‌ها آغاز کند، حتی اگر هنوز مشغول اجرای عملیات قبلی باشد.

لیست دستورالعمل‌ها

- **neg:**
  این دستور به صورت کلمه‌به‌کلمه روی بلوک src1 حرکت کرده و منفی هر کلمه را در موقعیت متناظر در بلوک dst ذخیره می‌کند.
- **sub:**
  این دستور به ترتیب از کلمات موجود در بلوک‌های src1 و src2 خوانده و حاصل تفریق آن‌ها را در موقعیت مشابه در بلوک dst قرار می‌دهد. به ازای هر اندیس i، مقدار src1[i] - src2[i] در dst[i] نوشته می‌شود.
- **add:**
  مشابه دستور sub، اما به جای تفریق، عملیات جمع انجام می‌شود. به ازای هر اندیس i، مقدار src1[i] + src2[i] در dst[i] ذخیره می‌گردد.

توجه: تمامی دستورات بالا باید به صورت تک‌چرخه‌ای پیاده‌سازی شوند.
گام دوم: در این گام، لازم است مجموعه‌ای از دستورات جدید به پردازنده طراحی‌شده اضافه شود تا بتواند با واحد محاسباتی نزدیک حافظه تعامل برقرار کند. این دستورات به پردازنده امکان می‌دهند مقادیر مورد نیاز را به واحد منتقل کرده، دستورالعمل لازم را تنظیم کند، عملیات را آغاز کرده و وضعیت انجام آن را بررسی کند.
در ادامه، لیست دستورات مورد نیاز به همراه توضیحات آن‌ها آورده شده است:

- **setsrc1 rs**
  مقدار موجود در ثبات rs را در ثبات src1 واحد محاسباتی قرار می‌دهد.
- **setsrc2 rs**
  مقدار موجود در ثبات rs را در ثبات src2 واحد محاسباتی قرار می‌دهد.
- **setdst rs**
  مقدار موجود در ثبات rs را در ثبات dst واحد محاسباتی قرار می‌دهد.

- **تنظیم دستورالعمل واحد محاسباتی**
برای مشخص کردن دستورالعمل مورد نظر (مانند add، sub یا neg) می‌توان از یک یا چند دستور مجزا استفاده کرد. اگر از پیشنهاد گام اول استفاده کرده‌اید و یک ثبات جداگانه برای نگهداری نوع دستور در نظر گرفته‌اید، می‌توانید دقیقاً مشابه دستورات setsrc1 و ... عمل کرده و مقدار لازم را در آن ثبات بنویسید.

  همچنین در صورت تمایل، می‌توانید برای هر نوع دستورالعمل، یک دستور مجزا تعریف کنید (مثلاً: setop_add، setop_neg و ...).

- **startnmc**
این دستور سیگنال شروع عملیات را به واحد محاسباتی ارسال می‌کند. با اجرای این دستور، واحد محاسباتی بر اساس مقادیر فعلی موجود در ثبات‌ها عملیات را آغاز خواهد کرد.

- **checkdone rt**
این دستور وضعیت اجرای واحد محاسباتی را بررسی می‌کند. اگر واحد مشغول انجام عملیات باشد (یعنی خروجی busy آن برابر ۱ باشد)، مقدار صفر در ثبات rt قرار می‌گیرد. در غیر این صورت (وقتی عملیات به پایان رسیده)، مقدار ۱ در rt ذخیره خواهد شد.

گام سوم:

در این گام، باید طراحی واحد محاسباتی به گونه‌ای اصلاح شود که اجرای آن هیچ تداخلی با عملکرد پردازنده اصلی ایجاد نکند. به طور خاص، هدف آن است که دسترسی به حافظه بین پردازنده و واحد محاسباتی به درستی هماهنگ شود.
برای این منظور، لازم است منطق کنترل دسترسی به حافظه به گونه‌ای تنظیم شود که:

- واحد محاسباتی فقط زمانی مجاز به خواندن از حافظه باشد که پردازنده در آن لحظه در حال خواندن داده از حافظه نیست.
- واحد محاسباتی تنها زمانی مجاز به نوشتن در حافظه باشد که پردازنده اصلی عملیات نوشتن انجام نمی‌دهد.

پیاده‌سازی این سیاست می‌تواند به صورت سیگنال‌های کنترلی مشترک انجام شود تا واحد محاسباتی پیش از اقدام به هرگونه دسترسی، وضعیت فعلی پردازنده را بررسی کند. بدین ترتیب، تداخل در دسترسی به حافظه میان دو واحد جلوگیری شده و عملکرد صحیح سیستم حفظ می‌شود.

قسمت امتیازی

در این بخش، واحد محاسباتی طراحی‌شده را به گونه‌ای گسترش دهید که از عملیات ضرب (mul) نیز پشتیبانی کند. این عملیات باید به صورت کلمه‌به‌کلمه بر روی بلوک‌های src1 و src2 اجرا شده و حاصل ضرب هر جفت کلمات در موقعیت متناظر در dst ذخیره شود:

$$\mathrm{dst}[i] = \mathrm{src۱}[i] \times \mathrm{src۲}[i]$$

نکته مهم در این بخش آن است که عملیات ضرب باید به صورت چندکلاکی (multi-cycle) پیاده‌سازی شود، به این معنا که انجام یک ضرب ممکن است در بیش از یک چرخه زمانی طول بکشد.

نحوه‌ی تحویل گزارش

- گام‌های پروژه را همراه با چالش‌های هنگام پیاده‌سازی و توضیحات هر بخش در گزارش خود توضیح دهید.
- برای ارزیابی عملکرد پروژه، لازم است یک برنامه ساده در دو حالت پیاده‌سازی و اجرا شود:
  - استفاده از ISA اولیه (بدون استفاده از واحد محاسباتی نزدیک حافظه)

- استفاده از ISA توسعه‌یافته با دستورات جدید مربوط به واحد محاسباتی

برنامه مورد نظر باید عملیات جمع دو ماتریس ۸×۸ را به صورت خانه‌به‌خانه انجام داده و نتیجه را در یک ماتریس خروجی ذخیره کند. در نهایت، مقدار نهایی این ماتریس خروجی در یک محل مشخص از حافظه نوشته شود.

با پیاده‌سازی صحیح واحد محاسباتی و استفاده مناسب از دستورات جدید در کد ارزیابی، انتظار می‌رود تعداد چرخه‌های زمانی مورد نیاز برای انجام عملیات کاهش یابد. این کاهش چرخه‌ها نشان‌دهنده‌ی افزایش کارایی به دلیل بهره‌گیری از محاسبات نزدیک حافظه خواهد بود.

- برای ارزیابی عملکرد واحد محاسباتی بخش امتیازی، لازم است هم با ISA اولیه (فاقد دستورات مخصوص واحد محاسباتی) و هم با ISA توسعه‌یافته (مجهز به دستور mul) برنامه‌ای پیاده‌سازی و اجرا شود.

  در این برنامه، باید موارد زیر انجام شوند:

  - ماتریس A با ۲ سطر و ۸ ستون، و ترانهاده‌ی ماتریس B با ۲ ستون و ۸ سطر، در حافظه قرار داده شوند.
  - حاصل ضرب این دو ماتریس محاسبه شود.
  - نتیجه در محل مشخصی از حافظه ذخیره گردد.

  پس از اجرای برنامه در هر دو حالت، باید تعداد چرخه‌های مورد نیاز برای انجام کامل محاسبه با یکدیگر مقایسه شوند. انتظار می‌رود استفاده از ISA جدید و واحد محاسباتی توسعه‌یافته منجر به کاهش چشمگیر در زمان اجرا و افزایش کارایی کلی سیستم شود.

پروژه نهم: طراحی سامانه پایش منابع کارت گرافیک (GPU Monitor)

مقدمه

با گسترش کاربردهای مبتنی بر یادگیری ماشین و پردازش‌های سنگین گرافیکی، استفاده از کارت‌های گرافیک [10] در مراکز داده و خوشه‌های محاسباتی به شدت افزایش یافته است. مدیریت درست این منابع نیازمند داشتن ابزارهای پایش دقیق برای بررسی مصرف GPU، حافظه، و فرآیندهای در حال اجراست. در نبود دسترسی به کارت‌های گرافیک فیزیکی، می‌توان از شبیه‌سازهایی مانند GPUSim یا اجرای nvidia-smi در حالت شبیه‌سازی (mock mode) برای آموزش و توسعه چنین ابزارهایی استفاده کرد.

صورت پروژه

هدف از این پروژه طراحی و پیاده‌سازی یک سامانه پایش منابع GPU است که بتواند وضعیت لحظه‌ای کارت‌های گرافیک را پایش کرده و نتایج را در قالبی ساخت‌یافته (مثلاً فایل JSON یا CSV) ذخیره کند.
این پروژه با استفاده از شبیه‌ساز یا ابزارهایی مانند nvidia-smi (در حالت شبیه‌سازی)، یا کتابخانه‌هایی نظیر pynvml یا gpustat انجام می‌گیرد.

گام‌های پروژه

- نصب و راه‌اندازی ابزار شبیه‌سازی مناسب برای محیط‌های فاقد GPU فیزیکی (مثلاً nvidia-smi در حالت mock یا استفاده از لاگ‌های مصنوعی).
- طراحی یک اسکریپت پایش که اطلاعات زیر را با فاصله‌های زمانی مشخص (مثلاً هر ۳۰ ثانیه) جمع‌آوری کرده و ذخیره کند:
  - میزان استفاده از GPU به درصد
  - حافظه استفاده‌شده از کارت گرافیک
  - فرآیندهای فعال بر روی GPU و شناسه‌های آن‌ها
  - دما، توان مصرفی، و وضعیت فن (در صورت موجود بودن در شبیه‌ساز)
- ذخیره‌سازی اطلاعات در فایل‌هایی با فرمت JSON یا CSV برای تحلیل‌های بعدی
- طراحی یک رابط خط فرمان ساده و داشبورد گرافیکی کوچک برای نمایش لحظه‌ای وضعیت کارت‌های گرافیک
- پیاده‌سازی قابلیت هشدار:
  - در صورت عبور دما از آستانه مشخص (مثلاً ۸۰ درجه) یک هشدار متنی نمایش داده شود
  - به صورت اختیاری، ارسال هشدار از طریق ایمیل یا پیام‌رسان[11]
- طراحی رابط کاربری گرافیکی ساده برای داشبورد پایش با استفاده از کتابخانه‌هایی نظیر matplotlib، plotly یا dash
- طراحی ساختار برنامه به‌گونه‌ای که توسعه آن برای چند GPU یا چند نود در آینده ممکن باشد (معماری مقیاس‌پذیر)

[10] GPU
[11] Messenger notification

گزارش و ارزیابی

- تمامی کدها، اسکریپت‌ها و فایل‌های پیکربندی باید به صورت کامل تحویل داده شوند.
- عملکرد درست سیستم باید با استفاده از لاگ‌ها و اجرای نمونه قابل مشاهده باشد.
- پروژه ترجیحاً باید بر روی مخزن GitHub توسعه یابد و تاریخچه commitها قابل مشاهده باشد (با ناظر پروژه برای ایجاد مخزن هماهنگ شود).
- در گزارش پروژه، روند طراحی سامانه، ابزارهای استفاده‌شده، چالش‌ها و نحوه حل آن‌ها به صورت دقیق مستند شود.
- نمونه‌ای از داده‌های پایش‌شده (حداقل ۱۰ نمونه زمانی) در گزارش ضمیمه شود.

بخش امتیازی

- توسعه خروجی وب برای مشاهده گزارش‌ها و وضعیت با استفاده از مرورگر
- طراحی ماژول ذخیره‌سازی اطلاعات در پایگاه‌داده SQLite یا MongoDB
- استفاده از ابزار پیشرفته DCGM[12] برای پایش سطح پایین منابع GPU و استخراج اطلاعات به‌روز از جمله:
  - وضعیت سلامت کارت گرافیک و تست‌های تشخیصی diagnostics
  - دما، توان مصرفی، بار پردازشی و حافظه
  - گزارش‌گیری از وضعیت کلی همه GPU ها (در صورت وجود)

و در صورت امکان، اتصال آن به داشبورد یا ذخیره‌سازی لاگ‌های خروجی برای تحلیل.

[12] Data Center GPU Manager

پروژه دهم: طراحی چارچوب محک کارت گرافیک (GPU Benchmarking Framework)

مقدمه

در یک سامانه دارای چند GPU یا حتی یک کارت گرافیک، دانستن توان عملیاتی واقعی آن نقش مهمی در برنامه‌ریزی و بهینه‌سازی استفاده از منابع دارد. محک به فرآیند اندازه‌گیری کارایی سیستم در انجام وظایف مشخص گفته می‌شود. این محک می‌تواند شامل بررسی سرعت انجام محاسبات ریاضی، مصرف حافظه، یا تأخیر در انتقال داده باشد. در این پروژه، هدف طراحی یک چارچوب محک کارت گرافیک است که بدون نیاز به GPU فیزیکی و با کمک شبیه‌ساز اجرا می‌شود.

صورت پروژه

در این پروژه، یک چارچوب محک ساده اما قابل توسعه طراحی و پیاده‌سازی خواهد شد که عملکرد GPU (یا محیط شبیه‌سازی‌شده) را در انجام عملیات خاص مورد ارزیابی قرار دهد. این عملیات می‌توانند شامل ضرب ماتریسی، انتقال داده بین حافظه‌ها، یا محاسبات تکراری ساده باشند. هدف، استخراج معیارهایی مانند زمان اجرا، نرخ استفاده از حافظه، و توان عملیاتی تقریبی است.

گام‌های پروژه

- طراحی و پیاده‌سازی یک یا چند عملیات محاسباتی ساده مانند ضرب ماتریس‌ها، جمع برداری یا محاسبات تکراری، با زبان Python یا C++.
- برای هر عملیات انتخاب‌شده (مانند ضرب ماتریس)، سه نسخه مختلف پیاده‌سازی کنید:
  - **نسخه اول:** پیاده‌سازی ساده و خطی آن عملیات فقط با استفاده از حلقه‌های تودرتو در Python یا C++، بدون استفاده از هیچ کتابخانه خاص. این نسخه نمایانگر عملکرد پایه بر روی CPU است.
  - **نسخه دوم:** پیاده‌سازی بهینه‌شده برای اجرای سریع‌تر روی CPU با استفاده از کتابخانه‌هایی مانند NumPy یا OpenMP، با هدف استفاده از قابلیت‌های موازی‌سازی داخلی پردازنده.
  - **نسخه سوم:** در صورت امکان، پیاده‌سازی نسخه‌ای که از شبیه‌سازی یا تسریع‌کننده‌های GPU استفاده کند. برای مثال، استفاده از numba.cuda یا سایر ابزارهایی که امکان نوشتن کرنل‌های ساده روی GPU را فراهم می‌کنند.
- اندازه‌گیری و ثبت زمان اجرا، میزان مصرف حافظه، و نرخ پردازش بر حسب عملیات بر ثانیه (FLOPS).
- ذخیره نتایج بنچمارک در قالب CSV یا JSON برای مقایسه‌های بعدی
- طراحی سیستم مقایسه‌ای که بتواند به‌صورت خودکار چند نسخه از الگوریتم‌ها را اجرا و مقایسه کند.
- طراحی رابط خط فرمان و داشبورد ساده برای مشاهده نتایج
- استفاده از کتابخانه matplotlib یا seaborn برای ترسیم نمودارهای حرفه‌ای از نتایج بنچمارک
- مقایسه عملکرد نسخه‌های مختلف و تحلیل علل تفاوت‌ها، به‌ویژه تأثیر نحوه دسترسی به حافظه یا ساختار حلقه‌ها

گزارش و ارزیابی

- کد کامل پروژه به همراه اسکریپت‌های اجرا باید تحویل داده شود.
- پروژه ترجیحاً باید بر روی مخزن GitHub توسعه یابد و تاریخچه commitها قابل مشاهده باشد (با ناظر پروژه برای ایجاد مخزن هماهنگ شود).
- نتایج محک به صورت فایل و نمودار در گزارش آورده شود و تحلیل معناداری از آن‌ها ارائه گردد.
- توضیح دهید که ساختار محک به گونه‌ای طراحی شده که در آینده بتوان عملیات یا الگوریتم‌های دیگر به آن افزود.

بخش امتیازی

- اضافه کردن یک بخش تحلیل آماری (میانگین، انحراف معیار، ...) برای تفسیر نتایج
- استفاده از ابزار پیشرفته DCGM[۱۳] برای مشاهده و تحلیل رفتار سیستم هنگام اجرای محک:
  - اجرای عملیات محک طراحی‌شده در پروژه به صورت همزمان با ابزار DCGM
  - استخراج و ثبت اطلاعاتی مانند:
    - میزان استفاده از حافظه GPU
    - میزان بهره‌وری[۱۴] در لحظات مختلف اجرا
    - تغییرات دما و توان مصرفی
  - تحلیل نحوه تغییر منابع در طول اجرای عملیات و بررسی تأثیر الگوریتم یا اندازه ورودی بر رفتار کارت گرافیک

[13] Data Center GPU Manager
[14] utilization