دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر
پروژه درس ساختار و زبان کامپیوتر

عنوان:

پروژه امتیازی سوم: موازی‌سازی با دستورات SIMD

Extra Credit Project 3: Parallel Computing Using SIMD

نگارش

گروه ۱:

سید احمد موسوی اول

سید امیرحسین موسوی فرد

رضا اسلامی ابیانه

آرمان طهماسبی زاده

استاد راهنما

دکتر حسین اسدی

بهمن ۱۴۰۳

فهرست مطالب

چکیده: در این سند خلاصه‌ای از اقدامات انجام شده در راستای پیاده‌سازی پروژه و نیز تمامی کدها و چالش‌های ایجاد شده در این فرایند همراه با توضیحات مربوطه قرار داده شده است.

واژه‌های کلیدی: موازی‌سازی، افزایش سرعت، SIMD، زبان C،زبان x86، تابع clock

۱ مقدمه

در این پروژه به بررسی تفاوت سرعت و عملکرد زبان اسمبلی x86 در مقابل زبان‌های سطح بالاتر مانند C و همچنین تاثیر موازی‌سازی در افزایش سرعت و عملکرد می‌پردازیم.

۱-۱ تعریف مسئله

در این مسئله باید در خط فرمان[۱] ابعاد دو ماتریس مربعی را دریافت کرده و سپس درایه‌های این دو ماتریس را دریافت کنیم. هدف این است که ضرب این دو ماتریس را در خط فرمان چاپ کنیم. این عملیات را باید یک بار با زبان C، یک بار با زبان اسمبلی x86 و یک بار با همین زبان اسمبلی و همچنین دستورات SIMD پیاده‌سازی کرده و زمان اجرا را برای هر سه پیاده‌سازی محاسبه کنیم. در نهایت نیز لازم است تا زمان‌های اجرا را روی نمودار رسم کرده و آن‌ها را مقایسه کنیم.

۱-۲ اهداف پژوهش

- آشنایی با دستورات SIMD و نحوه کار با آن‌ها.
- بهبود سرعت کد با استفاده از این دستورها و ثبات‌ها.

۱-۳ موارد مورد استفاده

زبان های برنامه نویسی C واسمبلی x86، فایل‌های مورد نیاز برای اجرای اسمبلی روی سیستم عامل لینوکس[۲] .زبان برنامه‌نویسی پایتون[۳] برای رسم نمودار و مقایسه داده‌های خروجی.

[۱] command line
[۲] linux
[۳] python

۲ پیاده‌سازی

در این بخش به توضیح نحوه‌ی ساخت این پروژه و رشد کردن آن شامل چالش‌های روبرو شده مانند گرفتن زمان اجرای هر زبان برای محاسبه پاسخ نهایی و موازی سازی می‌پردازیم.

۲-۱ توضیحات ابتدایی

در این پروژه بعد از تعریف پروژه ابتدا پروژه را به چندین بخش ساده‌تر تقسیم کردیم؛ در بخش اول ابتدا کد پروژه را به زبان C نوشتیم سپس کد عادی آن را به زبان x86 طراحی کردیم که این پیاده‌سازی بسیار ساده عمل می‌کرد و ضرب عادی دو ماتریس را با پیچیدگی $O(n^3)$ انجام می‌داد؛ در نهایت در بخش بعدی این پروژه را با استفاده از سری دستورات SIMD به صورت موازی‌سازی زدیم تا به پیچیدگی کمتری برسیم و زمان اجرای آن کمتر شود؛ در آخر نیز بعد از این قسمت، زمان‌های اجرای کدها را با استفاده از تابع clock که در زبان C پیاده‌سازی شده است به دست آورده و آن‌ها را با استفاده از زبان برنامه‌نویسی پایتون به نمودار تبدیل کردیم که در بخش نتایج آن‌ها را آورده‌ایم.

۲-۲ مراحل پیاده‌سازی

در مرحله اول این الگوریتم را در زبان C پیاده‌سازی می‌کنیم. در این مرحله ابتدا با تابع[۴] scanf اندازه ماتریس‌های ورودی را در خط اول دریافت کرده و در خطوط بعدی درایه‌ها را دریافت می‌کنیم؛ سپس برای محاسبه ماتریس ضرب این دو، یک حلقه بر روی سطرهای ماتریس اولی ، سپس یک حلقه بر روی ستون‌های ماتریس دومی و سپس یک حلقه برای پیمایش این سطر و ستون و ضرب درایه‌های آن، جمع کردن آن‌ها و سپس ذخیره در خانه مربوطه به ماتریس خروجی پیاده‌سازی می‌کنیم. سپس جواب نهایی ماتریس را در خط فرمان چاپ می‌کنیم. حال برای تغییر این کد به گونه‌ای که زمان متوسط اجرا شدن کد را به ما خروجی دهد ابتدا تابع بالا را برای ورودی‌ها به تعداد مشخصی (برای مثال هزار بار) در یک حلقه صدا می‌زنیم و با استفاده از تابع clock اختلاف زمان مورد نیاز برای اجرا شدن این حلقه را حساب می‌کنیم سپس بر تعداد دفعات اجرا تقسیم کرده تا به میانگین برسیم و آن را چاپ می‌کنیم.

در مرحله دوم از زبان اسمبلی x86 استفاده شده است؛ دلیل استفاده از این زبان برای پیاده سازی الگوریتمی مشابه بالا این است که در اسمبلی کنترل دقیق‌تری روی حافظه و ثبات‌ها داریم که باعث

[۴] function

افزایش سرعت می‌شود؛ الگوریتم پیاده‌سازی مشابه بخش قبل است با این تفاوت که برای اندازه‌گیری زمان از همان تابع clock استفاده می‌کنیم و مشابه بخش قبل به تعداد مشخصی الگوریتم را اجرا می‌کنیم که مدت زمان میانگین برای اجرای الگوریتم را به دست آوریم.
در مرحله سوم از زبان اسمبلی x86 و سری دستورات SIMD برای موازی‌سازی ضرب سطر ماتریس اول در ستون ماتریس دوم استفاده می‌کنیم به این‌گونه که دستورات SIMD همزمان ۴ عملیات ضرب را به صورت موازی انجام می‌دهند و باعث افزایش سرعت می‌شوند. در این بخش برای سازگاری با دستورات SIMD ابتدا باید ماتریس دوم را در یک تابع به صورت ترانهاده[۵] در آوریم تا انتقال داده‌ها به ثبات‌های SIMD راحت‌تر شود.

۲-۳ بررسی منطق کد در x86

ابتدا یک تابع به نام get_input برای ورودی گرفتن ماتریس‌های ورودی، نوشته شده است به طوری که ابتدا یک آرایه یک بعدی به طول $n \times n \times 8$ برای قرار دادن اعضای ماتریس در آن می‌سازیم.
سپس باید ۲ ماتریس را در همدیگر ضرب کنیم که برای آن تابع multiply_matrix زده شده است. در این تابع در ماتریس اول روی سطرهای آن یک حلقه زده شده است و در حلقه درونی آن روی ستون‌های ماتریس دومی یک حلقه زده شده است. حال برای محاسبه یک درایه در ماتریس نهایی درون این ۲ حلقه یک حلقه دیگر وجود دارد که اعضای یک سطر در ماتریس اول را در یک ستون ماتریس دومی ضرب می‌کند و در ماتریس نهایی قرار می‌دهد.
چون هر ماتریس به شکل یک آرایه تک بعدی ذخیره شده است در نتیجه برای اینکه بخواهیم به سطر بعدی یک ماتریس برویم به اندازه $n \times 8$ باید در آرایه جلو برویم.
در نهایت باید ماتریس نهایی را چاپ کنیم که با استفاده از تابع print_output این کار نیز انجام شده است.

۲-۴ چگونگی کار SIMD

```
calculate_element:
movdqu xmm0,[r11]
movdqu xmm1,[r12]
pmulld xmm0,xmm1
```

[۵] Transpose

```
phaddd xmm0, xmm0
phaddd xmm0, xmm0
movd ebx,xmm0
add edx,ebx
add r11,16
add r12,16
loop calculate_element
```

در لوپ بالا از دستورات SIMD برای موازی‌سازی ضرب دو سطر از ماتریس‌ها استفاده شده است. در این لوپ ابتدا مقادیر با استفاده از دستور movdqu به صورت چهارتایی در ثبات قرار می‌گیرند و سپس با لرpmulld به صورت موازی در هم ضرب می‌شوند. حال باید جمع مقادیر ضرب شده را محاسبه کنیم که با استفاده از دستور phaddd که دو بار اجرا شود جمع هر ۴ مقدار در اولین بخش xmm0 ذخیره می‌کند. حال می‌توانیم با استفاده از movd این مقدار را در ثبات مورد نظر ریخته و آن را با جمع باقی درایه‌ها که در edx ذخیره شده بود جمع بزنیم. حالا باید ۱۶ واحد به ثبات‌ها اضافه کنیم چون الان ضرب چهار عدد که هر کدام چهار بایت هستند را محاسبه کردیم. سپس لوپ را مجددا اجرا می‌کنیم تا تمامی اعداد متناظر دو سطر ضرب شوند.
برای ضرب ۲ ماتریس باید سطر در یک ماتریس را در ستون ماتریس دیگر ضرب کنیم و در ماتریس نهایی قرار دهیم. اما به دلیل استفاده از موازی‌سازی باید ۴ عضو متوالی از سطر ماتریس اول را در xmm0 و ۴ عضو متوالی از ستون ماتریس دوم را در xmm1 قرار دهیم و سپس از دستورات SIMD استفاده کنیم. و چون ماتریس‌ها به شکل یک آرایه یک بعدی ذخیره شده‌اند نمی‌توان از ماتریس دومی ۴ عضو متوالی در یک ستون را خواند و در ثبات مد نظر قرار داد. به همین دلیل ماتریس دوم را ترانهاده می‌کنیم و برای ضرب، یک سطر در ماتریس اول را در یک سطر در ماتریس دوم ضرب می‌کنیم. با این کار برای خواندن ۴ عضو متوالی از ستون ماتریس دوم کافی‌ست ۴ عضو متوالی در سطر متناظر ماتریس ترانهاده آن خوانده شود و در ثبات قرار داده شود.

۵-۲ چالش‌های پیاده‌سازی

بخش اول پروژه که با زبان C پیاده‌سازی شد بدون چالش خاصی به پایان رسید ولی در بخش دوم که باید با استفاده از زبان x86 کد را پیاده‌سازی می‌کردیم مشکلات و چالش‌های زیادی به همراه داشت؛ از این مشکلات می‌توان به موارد زیر اشاره کرد:

- **مدیریت بهینه اندازه پشته:** در این مورد یکی از مشکلات این بود که برای اعداد ماتریس‌ها فضای کافی در نظر گرفته نشده بود و اعداد به خوبی ذخیره نمی‌شدند؛ به همین دلیل خروجی‌ها با اعداد خواسته‌شده مطابقت نداشتند.

- **کار با SIMD:** به دلیل اینکه با این دستورات آشنا نبودیم؛ ابتدا با استفاده از جست‌وجو در اینترنت این دستورات را پیدا کردیم و به مطالعه آن‌ها و نحوه‌ی کار با آن‌ها پرداختیم تا بتوانیم از آن‌ها استفاده کنیم و یکی از چالش‌ها مخصوصاً در بخش سوم پروژه که باید از سری دستورات SIMD استفاده می‌کردیم؛ همین ناآشنا بودن با این دستورات بود.

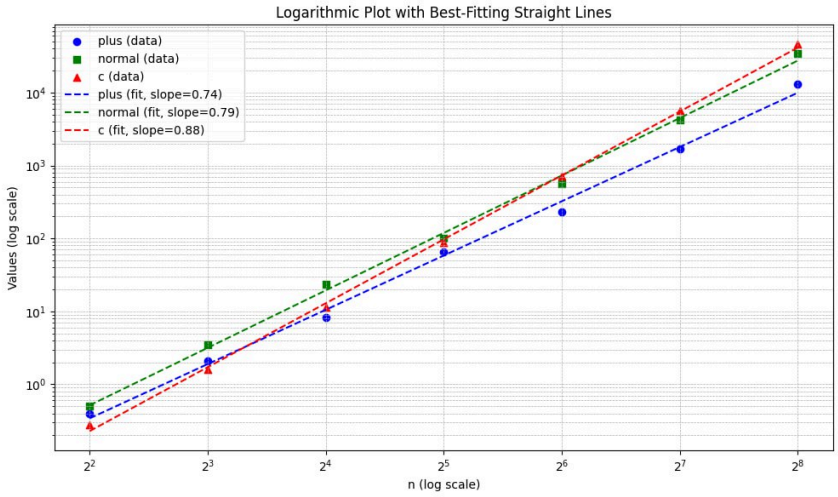
- **کار با تابع clock در زبان x86:** خروجی این تابع از نوع صحیح بود به همین دلیل نمی‌توانستیم آنها را در همان کد به اعداد اعشاری تبدیل کنیم و بنابراین آنها را به همان صورت صحیح خروجی گرفتیم؛ علاوه بر این یکی از چالش‌های دیگر تفاوت پیاده‌سازی این تابع در سیستم‌عامل linux و سیستم‌عامل windows بود که به همین دلیل به اعداد غیر قابل قبولی رسیدیم و متوجه شدیم که در windows این اعداد به صورت میلی‌ثانیه خروجی داده می‌شدند ولی در linux به صورت میکروثانیه خروجی داده می‌شدند.

۳ جمع‌بندی و خروجی نهایی

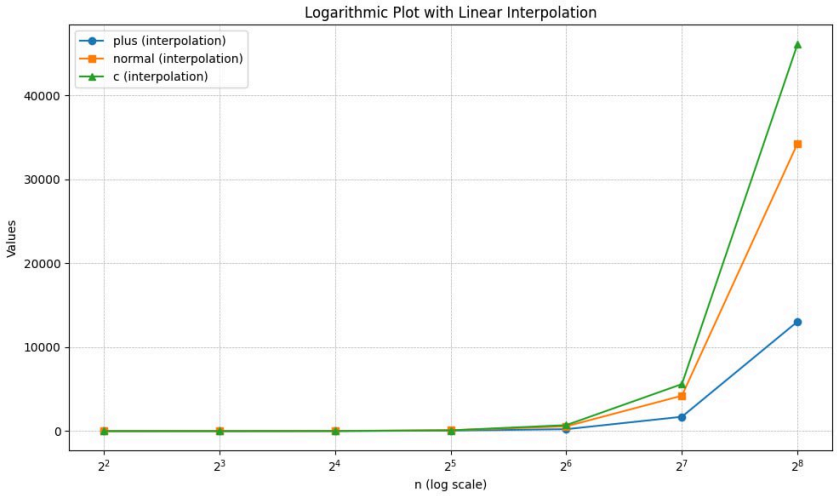
در نهایت هر یک از سه کد به ما سه خروجی می‌دهند که به غیر از خروجی ماتریس نهایی هر کدام هم یک عدد متناسب با زمان اجرای الگوریتم می‌دهند. در کد C خروجی زمانی میانگین زمانی است که هر ضرب ماتریس در واحد میلی‌ثانیه طول کشیده است ولی خروجی زمانی دو کد اسمبلی x86 مدت زمان کل اجرا شدن هزار مرتبه از ضرب ماتریس را با واحد میکروثانیه می‌دهد. به همین علت دو خروجی با اردر $۱۰^۶$ با یکدیگر تفاوت دارند و برای تبدیل باید در این عدد ضرب شوند.
پس از خروجی گرفتن از هر یک از سه کد با استفاده از فایل های input داده‌ها را بدست آورده و در فایل data.txt ذخیره می‌کنیم. سپس با استفاده از اسکریپت پایتون به نام graph_maker استفاده می‌کنیم و دو نمودار خطی و لگاریتمی به همراه رگرسیون خطی آن‌ها را بدست می‌آوریم.

طبق نمودار خطی می‌بینیم که در ماتریس ها با اندازه بزرگتر کد بهینه با استفاده از سری دستورات SIMD در مقابل دستورات عادی زبان x86 و زبان سطح بالای c بین سه تا چهار برابر سریع‌تر عمل می‌کند. همچنین در نمودار لگاریتمی که برازش خطی روی انها انجام شده است می‌بینیم که شیب مربوطه به مدت زمان و اندازه ورودی به ترتیب در کد بهینه با سری دستورات SIMD ، کد اسمبلی

x86 و کد با استفاده از زبان C کمتر است. این به آن معناست که با افزایش اندازه ورودی ، سرعت رشد زمان لازم برای اجرای الگوریتم برای کد c و کد غیر بهینه اسمبلی x86 بیشتر از کد بهینه اند که به این معناست این کد ها با افزایش اندازه ورودی به مراتب (چون برازش لگاریتمی است) کند تر عمل می‌کنند تا کد بهینه با سری دستورات SIMD.



شکل ۱: نمودار لگاریتمی مقایسه‌ی زمان‌ها



شکل ۲: نمودار خطی مقایسه‌ی زمان‌ها