بهبود کارایی الگوریتم هم‌ترازسازی اسمیت-واترمن با استفاده از واحد پردازنده‌ی گرافیکی

نرجس شاهمرادی و اسدالله شاه‌بهرامی

گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه گیلان، nshahmoradi@webmail.guilan.ac.ir
shahbahrami@guilan.ac.ir

چکیده

هم‌ترازسازی توالی‌های بیولوژیکی یکی از وظایف مهم و چالش‌برانگیز بیوانفورماتیک است. این فرایند برای تحلیل و شناسایی نواحی مشابهت بین توالی‌ها به کار می‌رود. الگوریتم‌های مختلفی برای چنین تحلیل‌هایی وجود دارند که از لحاظ دقت و پیچیدگی محاسباتی با یکدیگر متفاوتند و در دو دسته‌ی کلی عمومی و محلی می‌گنجند. الگوریتم اسمیت-واترمن یکی از دقیق‌ترین الگوریتم‌های مبتنی بر برنامه‌نویسی پویا برای یافتن بهترین هم‌ترازسازی محلی بین دو توالی است. این الگوریتم در کنار دقت بالایی که دارد، دارای پیچیدگی محاسباتی متناسب با حاصل‌ضرب طول دو توالی است. به همین دلیل، در زمره‌ی الگوریتم‌های زمان‌بر است و برای افزایش سرعت آن از موازی‌سازی استفاده خواهد شد. هدف این مقاله، افزایش کارایی الگوریتم اسمیت-واترمن با استفاده از تکنیک موازی‌سازی بر روی GPU است. مرحله‌ی محاسبه‌ی ماتریس امتیازدهی بر روی GPU پیاده‌سازی شده است. بررسی‌های انجام گرفته نشان می‌دهند که این پیاده‌سازی موازی از الگوریتم اسمیت-واترمن، تسریع ۴۷ برابری نسبت به حالت پیاده‌سازی سریال الگوریتم دارد.

واژه‌های کلیدی

الگوریتم اسمیت-واترمن، برنامه‌نویسی پویا، بیوانفورماتیک، موازی‌سازی، هم‌ترازسازی توالی.

۱- مقدمه

هم‌ترازسازی توالی در بیوانفورماتیک روشی برای مرتب‌سازی توالی‌های بیولوژیکی نسبت به یکدیگر است به گونه‌ای که نواحی مشابهت بین آنها شناسایی شود. این نواحی مشابهت نتیجه‌ی روابط ساختاری، عملکردی یا تکاملی بین توالی‌ها است. با هم‌تراز کردن یک توالی ناشناخته با سایر توالی‌های شناخته شده‌ی موجود در یک پایگاه‌داده می‌توان به ویژگی‌های توالی ناشناخته دست یافت [۱]. دو نوع هم‌ترازسازی وجود دارد که عمومی و محلی نام دارند. هم‌ترازسازی عمومی شباهت را در سرتاسر توالی‌ها پیدا می‌کند. در حالی که هم‌ترازسازی محلی زیرتوالی‌های با بیشترین شباهت بین توالی‌ها را شناسایی می‌کند. الگوریتم نیدلمن-وانچ برای به دست آوردن هم‌ترازسازی عمومی و الگوریتم‌های اسمیت-واترمن، FASTA و BLAST برای به دست آوردن هم‌ترازسازی محلی پیشنهاد داده شده‌اند [۲]. الگوریتم اسمیت-واترمن یکی از دقیق‌ترین الگوریتم‌های مبتنی بر برنامه‌نویسی پویا است که پیچیدگی محاسباتی متناسب با حاصلضرب طول دو توالی دارد و میزان زیادی از محاسبات را هنگام جستجوی پایگاه‌داده‌های توالی‌های بیولوژیکی متحمل می‌شود [۳]. به عنوان مثال الگوریتم BLAST نسبت به اسمیت-واترمن تا ۴۰ برابر سریع‌تر است اما دقت نتایج آن به اندازه‌ی دقت نتایج حاصل از اسمیت-واترمن نیست. بنابراین نیاز به اعمال روش‌هایی برای تسریع الگوریتم اسمیت-واترمن است [۴].

سکوی GPU، یک سکوی تسریع‌دهنده‌ی نسبتاً کم هزینه و کارآمد برای هم‌ترازسازی توالی‌های بیولوژیکی است. در زمینه‌ی بهبود کارایی الگوریتم اسمیت-واترمن، تلاش‌های زیادی برای پیاده‌سازی آن بر روی GPU انجام شده است. نمونه‌هایی از این پیاده‌سازی‌ها در [۴-۱۳] نشان داده شده‌اند که بهبود سرعت در آنها به دست آمده است ولی یک مقایسه‌ی مناسبی از پیاده‌سازی موازی اسمیت-واترمن بر روی چندین GPU انجام نشده است.

هدف این مقاله، افزایش کارایی الگوریتم اسمیت-واترمن بر روی سکوی GPU است که برای این کار، محاسبه‌ی ماتریس امتیازدهی به عنوان مرحله‌ی زمان‌بر در الگوریتم بر روی GPU پیاده‌سازی شده است. در این پیاده‌سازی، هر نخ، مسئولیت محاسبه‌ی ماتریس امتیازدهی یک هم‌ترازسازی را برعهده دارد. بنابراین می‌توان با بکارگیری نخ‌های مستقل از هم چندین هم‌ترازسازی را به موازات هم انجام داد. نتایج پیاده‌سازی بر روی یک GPU برای دو توالی با طول‌های متفاوت نشان داده که کارایی پیاده‌سازی موازی نسبت به سریال تا حدود ۴۷ برابر است.

این مقاله به صورت زیر سازمان‌دهی می‌شود. الگوریتم هم‌ترازسازی محلی اسمیت-واترمن در قسمت دوم بررسی می‌شود. قسمت سوم به شرح موازی‌سازی الگوریتم اسمیت-واترمن پرداخته و نتایج مربوط به پیاده‌سازی آن در بخش چهار توضیح داده خواهد شد. سپس در بخش پنجم شرح مختصری از کارهای مرتبط بیان می‌شود و بخش آخر در برگیرنده‌ی نتیجه مقاله خواهد بود.

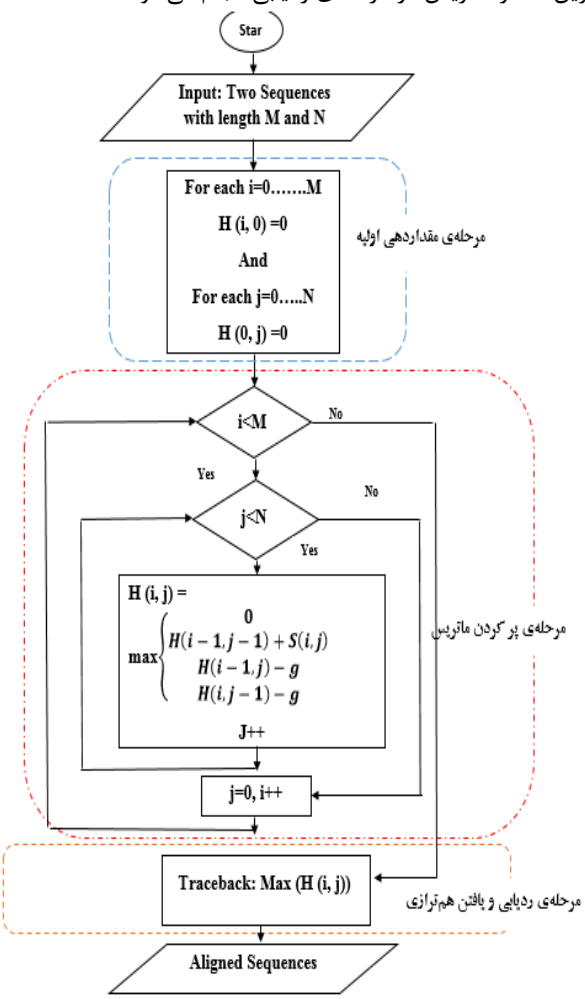
۲- الگوریتم اسمیت-واترمن

از توالی‌هایی که به طور کلی شباهت کمی به هم دارند اما زیرتوالی‌های مشابهی در آنها وجود دارد، ارتباطات مهمی را می‌توان استخراج کرد. الگوریتم اسمیت-واترمن یک روش دقیق است که با به دست آوردن هم‌ترازسازی محلی بهینه بین دو توالی، زیرتوالی‌های مشابه بین آنها را شناسایی می‌کند. هدف این الگوریتم، یافتن عناصر یک ماتریس شباهت به نام H و به دست آوردن هم‌ترازسازی محلی بهینه از روی همین ماتریس است. ابتدا توالی‌های A و B به ترتیب با ابعاد m و n به عنوان ورودی دریافت می‌شوند و ماتریس H با ابعاد (n+1)×(m+1) ساخته می‌شود. H(i,j)، امتیاز شباهت زیرتوالی‌های A[1…i] و B[1…j] از دو توالی A و B را نشان می‌دهد به طوری که $1 \le i \le m$ و $1 \le j \le n$ است. در ابتدا اولین سطر و ستون از ماتریس H با صفر مقداردهی می‌شوند، سپس سایر عناصر ماتریس، طبق رابطه ۱ محاسبه می‌شوند [۱۴].

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S(a_i, b_j) \approx match/mismatch \\ H_{i-1,j} - g(a_i, -) \approx deletion \\ H_{i,j-1} - g(-, b_j) \approx insertion \end{cases} \quad (۱)$$

همانطور که از رابطه‌ی ۱ مشخص است برای به دست آوردن امتیازهای ماتریس از یک ماتریس جانشینی به نام S و یک تابع جریمه‌ی پرش نیز استفاده می‌شود. برای توالی‌های پروتئینی معمولاً از ماتریس‌های جانشینی BLOSUM یا PAM استفاده می‌شود که عناصر این ماتریس‌ها، احتمال جایگزینی دو کاراکتر بیولوژیکی با یکدیگر را نشان می‌دهند. تابع g برای جریمه‌ی پرش‌هایی است که ممکن است در توالی‌ها ایجاد شوند. با جستجوی کل ماتریس H و یافتن بزرگترین عنصر آن، بهترین مشابهت محلی بین دو توالی به دست می‌آید. پیچیدگی زمانی این مرحله از الگوریتم برابر با O(mn) است [۱۵].

برای به دست آوردن هم‌ترازی، بعد از یافتن عنصر با مقدار ماکزیمم، از همین عنصر شروع کرده و به عقب بر می‌گردد تا به عنصر اول ماتریس یعنی H(0,0) برسد. اما در این مسیر هر جا که به مقدار صفر برخورد کند، متوقف شده و همان نقطه، شروع هم‌ترازی خواهد بود. بنابراین می‌توان هم‌ترازی بهینه را با دنبال کردن این مسیر که هر گام آن متناظر با یک جانشینی، درج یا حذف است، به دست آورد. جهش افقی، قطری و عمودی به ترتیب به معنای ایجاد پرش در توالی A، تطابق و ایجاد پرش در توالی B است. مراحل الگوریتم اسمیت-واترمن در سه مرحله به صورت فلوچارت در شکل ۱ نشان داده شده است. مرحله‌ی مقداردهی اولیه مربوط به پر کردن سطر و ستون اول ماتریس، مرحله‌ی پر کردن ماتریس مرتبط با پر کردن عناصر ماتریس امتیازدهی است. به دست آوردن هم‌ترازسازی از روی بزرگترین مقدار ماتریس در مرحله‌ی ردیابی انجام می‌شود [۱۶].



شکل ۱: فلوچارت الگوریتم اسمیت-واترمن

رابطه‌ی ۱ طبق جریمه‌ی پرش خطی است. اگر از جریمه‌ی پرش نسبی در الگوریتم اسمیت-واترمن استفاده شود، ماتریس H طبق روابط ۲، ۳ و ۴ محاسبه می‌شود. مقادیر α و β به ترتیب جریمه‌ی آغاز پرش و جریمه‌ی گسترش پرش را نشان می‌دهند [۱۰].

$$H_{i,j} = \max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + S(a_i, b_j) \end{cases} \quad (۲)$$

$$E_{i,j} = \max \begin{cases} H_{i,j-1} - \alpha \\ E_{i,j-1} - \beta \end{cases} \quad (۳)$$

$$F_{i,j} = \max \begin{cases} H_{i-1,j} - \alpha \\ F_{i-1,j} - \beta \end{cases} \quad (۴)$$

یک نمونه هم‌ترازسازی محلی برای دو توالی GTCTATCAC و ATCTCGTATGAT با الگوریتم اسمیت-واترمن در شکل ۲ نشان داده شده است. در این مثال جریمه‌ی پرش خطی، امتیاز تطبیق و امتیاز عدم تطبیق به ترتیب ۱، ۲ و ۱- فرض شده است. بزرگترین امتیاز ماتریس عدد ۱۰ است که به دست آوردن هم‌ترازی بهینه از همین عنصر شروع می‌شود.

		#	A	T	C	T	C	G	T	A	T	G	A	T
	#	0	0	0	0	0	0	0	0	0	0	0	0	0
A	G	0	0	0	0	0	0	2	1	0	0	2	1	0
	T	0	0	2	1	2	1	1	4	3	2	1	1	3
	C	0	0	1	4	3	4	3	3	3	2	1	0	2
	T	0	0	2	3	6	5	4	5	4	5	4	3	2
	A	0	2	2	2	5	5	4	4	7	6	5	6	5
	T	0	1	4	3	4	4	4	6	5	9	8	7	8
	C	0	0	3	6	5	6	5	5	5	8	8	7	7
	A	0	2	2	5	5	5	5	4	7	7	7	10	9
	C	0	1	1	4	4	7	6	5	6	6	6	9	9

B

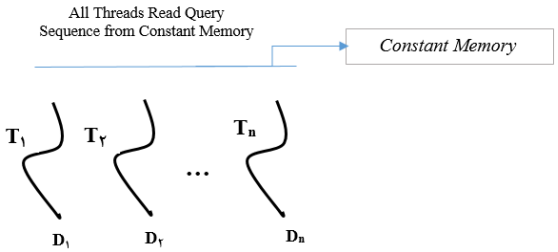
A: TC-TATCA

B: TCGTATGA

شکل ۲: محاسبه‌ی هم‌ترازسازی محلی بهینه بین دو توالی

۳- موازی‌سازی الگوریتم اسمیت-واترمن

در پیاده‌سازی انجام شده، هر کدام از هسته‌های پردازشی روی GPU به طور مستقل، کد هسته را بر روی مجموعه داده‌هایی که در اختیار آنها قرار گرفته، اجرا می‌کنند. کد هسته حاوی دستورات الگوریتم اسمیت-واترمن است و فقط محاسبه‌ی ماتریس هم‌ترازسازی را بر روی GPU انجام می‌دهد. در واقع استراتژی پیکربندی نخ‌ها برای اجرا روی چندپردازنده‌های GPU به این صورت است که هنگام جستجو در بانک‌های اطلاعاتی، هر نخ هم‌ترازسازی توالی مورد جستجو با یکی از توالی‌های پایگاه‌داده را انجام دهد. GPU دارای انواع مختلف حافظه است که هر کدام برای استفاده‌های مختلفی بهینه‌سازی شده‌اند. حافظه‌ی ثابت یکی از انواع حافظه GPU است که اگر همه‌ی نخ‌های اجرایی، داده‌ها را از آدرس یکسانی در این حافظه بخوانند، سرعت خواندن از آن تقریباً معادل با سرعت خواندن از ثبات‌ها خواهد بود [۱۷]. از طرفی چون الگوریتم اسمیت-واترمن برای هم‌تراز کردن یک توالی مورد جستجو با هر کدام از توالی‌های موجود در یک بانک اطلاعاتی به کار می‌رود، در این پیاده‌سازی توالی مورد جستجو در حافظه ثابت ذخیره شده است تا همه‌ی نخ‌ها برای انجام محاسبات خود یک توالی مورد جستجو را از یک آدرس مشخص بخوانند. شکل ۳ اجرای همزمان چندین هم‌ترازسازی را نشان می‌دهد.



شکل ۳: اجرای همزمان نخ‌ها برای هم‌ترازسازی یک توالی پرس‌وجو با توالی‌های مختلف پایگاه‌داده

هسته‌های پردازشی GPU به صورت چندپردازنده‌ها دسته‌بندی شده‌اند و هر کدام از چندپردازنده‌ها شامل مجموعه‌ای از پردازنده‌های جریانی هستند. نخ‌ها برای اجرای روی چندپردازنده‌ها به صورت سلسله‌مراتبی در بلاک‌ها و توری‌ها دسته‌بندی می‌شوند که هر بلاک شامل مجموعه‌ای از نخ‌ها و هر توری شامل مجموعه‌ای از بلاک‌های مستقل از هم خواهد بود. به همین منظور در پیاده‌سازی انجام شده هنگام جستجو در بانک اطلاعاتی پیکربندی نخ‌ها به این صورت است که به ازای هر بلاک، ۶۴ نخ در نظر گرفته شده و تعداد بلاک‌های هر توری با توجه به تعداد کل توالی‌های موجود در بانک اطلاعاتی محاسبه خواهد شد.

هنگام جستجو در بانک‌های اطلاعاتی، توالی‌های آن بر حسب طولشان مرتب‌سازی می‌شوند. چون اجرای نخ‌ها روی هر چندپردازنده به صورت ریسمان‌های ۳۲تایی است و اگر زمان خاتمه‌ی همه نخ‌های ریسمان نزدیک به هم باشد، می‌توان هزینه‌ی همزمان‌سازی نخ‌ها را کاهش داد. محاسبات ماتریس به صورت ستونی انجام خواهد شد و در هر لحظه، چهار سلول ماتریس محاسبه می‌شود. مکانی موقتی در حافظه‌ی عمومی نیز در نظر گرفته شده است تا مقادیری را که به خاطر وابستگی‌های داده‌ای بین سلول‌های ماتریس در محاسبه‌ی ستون بعدی به آنها نیاز خواهد بود، نگهداری کند. یعنی هر بار که یک ستون در ماتریس محاسبه شود، مقادیر آن در حافظه‌ی موقتی قرار خواهد گرفت و این حافظه با محاسبه‌ی ستون جدید بروز رسانی می‌شود تا مقادیر لازم برای ستون بعدی را در اختیار آن قرار دهد.

برای هر کدام از نخ‌ها، ثباتی در نظر گرفته می‌شود تا بزرگترین امتیاز هم‌ترازسازی را نگهداری کند و این ثبات در هر مرحله به روزرسانی می‌شود تا زمانی که هم‌ترازسازی کامل شود. در پایان هم‌ترازسازی، مقدار ثبات در حافظه‌ی عمومی نوشته خواهد شد. کل ماتریس در حافظه ذخیره نمی‌شود و فقط مقادیر موقتی که به خاطر وابستگی‌های داده‌ای به آنها نیاز خواهد بود، نگهداری خواهد شد. چهار مقدار موقتی در ابتدای محاسبات ستون جدید خوانده می‌شوند و مقادیر جدید محاسبه شده برای محاسبه‌ی ستون بعدی که آنها را نیاز خواهد داشت، در انتها نوشته می‌شوند. شکل ۴ شبه کد پیاده‌سازی انجام شده را نشان می‌دهد.

```
main()
{
  allocate memory on host for refrence sequences
  allocate memory on device for refrence sequences
  copy the Sequences from host to device
  allocate memory on host and device for Scores and Temprary Values
  Launch kernel
   {
     for (each character of refrence sequence)
          {
             for (each character of query sequence)
              {
                for (each of 4 character block)
                  {
                    find maxscore between (0,
                                           H(i-1,j-1)+S(i,j)
                                           H(i-1,j)-gap penalty
                                           H(i,j-1)-gap penalty)
                  }
              }
          }

  Transfer Result from device memory to host memory
}
```

شکل ۴: شبه کد الگوریتم اسمیت-واترمن موازی

برای امتیازدهی به تطبیق و عدم تطبیق کاراکترهای بیولوژیکی از ماتریس جانشینی BLOSUM در این پیاده‌سازی استفاده شده است. برای این که دسترسی سریع و بهینه‌ای به عناصر این ماتریس انجام گیرد، تغییراتی روی این ماتریس اعمال می‌شود. از آنجا که عناصر این ماتریس استاندارد، کاراکترهای الفبا هستند، ابتدا به ترتیب حروف الفبا مرتب می‌شوند و سپس تعدادی کاراکتر تهی در مکان‌هایی که حروف الفبا استفاده نشده است، اضافه می‌شوند. نوع ماتریس جانشینی هیچ ارتباطی به این روش ندارد، تا زمانی که در ماتریس، ۲۶ حرف الفبا استفاده شود و به ترتیب حروف الفبا مرتب‌سازی شده باشند. با بکارگیری این نوع روش دسترسی به ماتریس جانشینی، علاوه بر دسترسی بهینه به مقادیر ماتریس جانشینی، فضای نسبتاً کمی از حافظه‌ی ثابت استفاده خواهد شد. چون حروف به ترتیب الفبا مرتب‌سازی شده‌اند اگر هنگام دسترسی به این ماتریس از کد اسکی مربوط به هر کاراکتر، عدد ۶۵ کسر شود مکان مورد نظر در ماتریس به دست می‌آید.

بنابراین با بکارگیری روش‌های پیشنهاد شده، پیچیدگی محاسباتی الگوریتم اسمیت-واترمن از O(mn) به O(mn/t) کاهش می‌یابد که m و n طول توالی‌هایی است که با هم، هم‌تراز می‌شوند و t نیز تعداد کل نخ‌هایی است که درون هسته راه‌اندازی خواهند شد.

۴- پیاده‌سازی و نتایج

الگوریتم مورد نظر در شرایط مختلف و روی کارت‌های گرافیکی مختلف مورد ارزیابی قرار گرفته است.

۴-۱- سیستم مورد استفاده

مشخصات سیستم‌های مختلف که الگوریتم موازی بر روی آنها اجرا شده در جدول ۱ و ویژگی‌های بیشتر در مورد GPUهای آنها در جدول ۲ نشان داده شده است.

جدول ۱: مشخصات سیستم‌های تحت بررسی

مشخصات سیستم	سیستم ۱	سیستم ۲
RAM	۴ گیگابایت	4 گیگابایت
CPU	Core 2 Duo 2.67 GHz	Core i5 1.80GHz
GPU	GeForce 9300M GS	GeForce GT 740M
Operating System	ویندوز هفت ۶۴ بیتی	ویندوز هفت ۶۴ بیتی

جدول ۲: مشخصات GPUها

Model	Launch	Architecture	Core	memory
GPU-1: GeForce 9300M GS	2008	Unified Shader Architecture	8	256 MB
GPU-2: GeForce GT 740M	2013	Kepler	384	2048 MB

۴-۲- داده‌های استفاده شده

توالی‌هایی که در این تحقیق بررسی می‌شوند، توالی‌های پروتئینی در قالب FASTA هستند. قالب FASTA یک قالب مبتنی بر متن برای نمایش توالی‌های آمینواسیدی به صورت دنباله‌ای از حروف است. یک توالی در قالب FASTA با یک توصیف یک خطی در مورد آن توالی آغاز می‌شود که این خط با یک علامت ”>“ در ابتدای آن شروع می‌شود. اولین کلمه‌ای که بعد از علامت ”>“ قرار می‌گیرد، شناسه‌ی توالی و بقیه موارد در آن خط، توضیحات مربوط به اطلاعات بیولوژیکی توالی هستند. نمونه‌ای از سه توالی تحت بررسی در این تحقیق به قالب FASTA در شکل ۵ نشان داده شده است که از پایگاه‌دادهی UniProt انتخاب شده‌اند و قالب استانداردی که مرکز ملی اطلاعات زیست فناوری، برای خط توصیف توالی‌های آن تعریف کرده است به صورت زیر است [۹]:

sp|accession|entry name

```
>sp|Q197F8|002R_IIV3 Uncharacterized protein 002R OS=Invertebrate iridescent virus 3 GN=IIV3-002R PE=4 SV=1
MASNTVSAQGGSNRPVRDFSNIQDVAQFLLFDPIWNEQPGSIVPWKMNREQALAERYPEL
QTSEPSEDYSGPVESLELLPLEIKLDIMQYLSWEQISWCKHPWLWTRWYKDNVVRVSAIT
FEDFQREYAFPEKIQEIHFTDTRAEEIKAILETTPNVTRLVIRRIDDMNYNTHGDLGLDD
LEFLTHLMVEDACGFTDFWAPSLTHLTIKNLDMHPRWFGPVMDGIKSMQSTLKYLYIFET
YGVNKPFVQWCTDNIETFYCTNSYRYENVPRPIYVWVLFQEDEWHGYRVEDNKFHRRYMY
STILHKRDTDWVENNPLKTPAQVEMYKFLLRISQLNRDGTGYESDSDPENEHFDDESFSS
GEEDSSDEDDPTWAPDSDDSDWETETEEEPSVAARILEKGKLTITNLMKSLGFKPKPKKI
QSIDRYFCSLDSNYNSEDEDFEYDSDSEDDDSDSEDDC
>sp|Q197F7|003L_IIV3 Uncharacterized protein 003L OS=Invertebrate iridescent virus 3 GN=IIV3-003L PE=4 SV=1
MYQAINPCPQSWYGSPQLEREIVCKMSGAPHYPNYYPVHPNALGGAWFDTSLNARSLTTT
PSLTTCTPPSLAACTPPTSLGMVDSPPHINPPRRIGTLCFDFGSAKSPQRCECVASDRPS
TTSNTAPDTYRLLITNSKTRKNNYGTCRLEPLTYGI
>sp|Q6GZX2|003R_FRG3G Uncharacterized protein 3R OS=Frog virus 3 (isolate Goorha) GN=FV3-003R PE=3 SV=1
MARPLLGKTSSVRRRLESLSACSIFFFLRKFCQKMASLVFLNSPVYQMSNILLTERRQVD
RAMGGSDDDGVMVVALSPSDFKTVLGSALLAVERDMVHVVPKYLQTPGILHDMLVLLTPI
FGEALSVDMSGATDVMVQQIATAGFVDVDPLHSSVSWKDNVSCPVALLAVSNAVRTMMGQ
PCQVTLIIDVGTQNILRDLVNLPVEMSGDLQVMAYTKDPLGKVPAVGVSVFDSGSVQKGD
AHSVGAPDGLVSFHTHPVSSAVELNYHAGWPSNVDMSSLLTMKNLMHVVVAEEGLWTMAR
TLSMQRLTKVLTDAEKDVMRAAAFNLFLPLNELRVMGTKDSNNKSLKTYFEVFETFTIGA
LMKHSGVTPTAFVDRRWLDNTIYHMGFIPWGRDMRFVVEYDLDGTNPFLNTVPTLMSVKR
KAKIQEMFDNMVSRMVTS
```

شکل ۵: قالب توالی‌های داده‌ای پایگاه‌داده به فرمت fasta

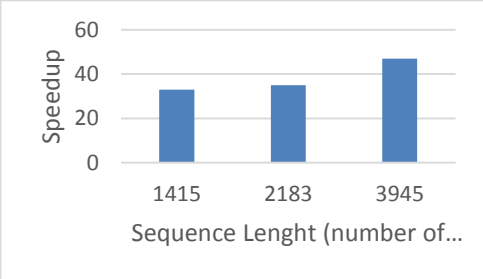
بانک‌های اطلاعاتی با تعداد توالی‌های مختلف برای آزمایش‌ها در نظر گرفته شده‌اند که توالی‌های آنها از UniProt انتخاب شده‌اند. مشخصات این بانک‌های اطلاعاتی در جدول ۳ نشان داده شده است.

جدول ۳: مشخصات بانک‌های اطلاعاتی مورد بررسی

بانک اطلاعاتی	DB_1	DB_2	DB_3	DB_4	DB_5
تعداد توالی‌ها	500	1000	1500	2000	2500

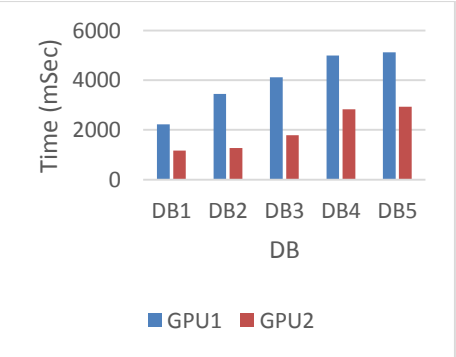
۴-۳- نتایج روی CPU و GPU

برای به دست آوردن کارایی الگوریتم اسمیت-واترمن بر روی GPU، این الگوریتم روی سیستم ۲ ذکر شده در بالا روی CPU و GPU آن پیاده‌سازی گردید. یک توالی با طول ۱۰۲۴ کاراکتر با توالی‌های مختلف با طول‌های ۱۴۱۵، ۲۱۸۳ و ۳۹۴۵ هم‌تراز گردید. کارایی پیاده‌سازی الگوریتم فوق روی GPU-2 نسبت به CPU در شکل ۶ نشان داده شده است. همانطور که از شکل پیداست با افزایش طول توالی‌ها، حجم محاسبات زیاد شده و کارایی GPU-2 نسبت به CPU افزایش می‌یابد که در این شکل ماکزیمم کارایی برابر با ۴۷ است. همانطور که قبلاً اشاره شده در موازی‌سازی این الگوریتم در هر نخ هر بار یک کاراکتر از توالی اول با چهار کاراکتر از توالی دوم مقایسه می‌شود و بعد از اتمام این کار دوباره چهار کاراکتر بعدی انتخاب می‌گردد.



شکل ۶: نمودار کارایی GPU نسبت به CPU

به منظور شناخت محدودیت‌های GPU-1، برنامه فوق روی مجموعه‌های مختلف جدول ۳ با یک توالی مورد جستجو به طول ۲۵۶ بر روی دو سیستم GPU-1 و GPU-2 با امکانات تعداد هسته‌های پردازشی و حافظه مختلف اجرا گردید. شکل ۷ زمان سپری شده روی دو GPU را نشان می‌دهد.



شکل ۷: زمان سپری شده برای جستجوی توالی ۲۵۶ کاراکتری درون بانک‌های اطلاعاتی مختلف بر روی دو سکوی GPU-1 و GPU-2

همانطور که انتظار می‌رفت، طبق شکل ۷ کارایی GPU-2 که دارای امکانات تعداد هسته‌های پردازشی و حافظه بیشتر است بالاتر از GPU-1 می‌باشد چرا که تعداد واحدهای پردازشی و اندازه حافظه در میزان کارایی نقش مهمی برای هم‌ترازسازی دارد.

۵- کارهای مرتبط

پیاده‌سازی‌های مختلفی از الگوریتم اسمیت-واترمن بر روی GPU انجام شده‌اند. اولین پیاده‌سازی الگوریتم اسمیت-واترمن روی GPU در [۶،۷] انجام شد. در این پیاده‌سازی‌ها از OpenGL برای اجرای الگوریتم روی GPU استفاده شده و اجرای الگوریتم مبتنی بر خط لوله‌ی گرافیکی بوده است. در [۴] اولین راه‌حل تسریع مبتنی بر CUDA معرفی شده است. در این کار برای اجتناب از جستجوی ماتریس جانشینی در حلقه‌ی داخلی الگوریتم و همچنین سرعت بخشیدن به دسترسی به آن از مفهوم پروفایل پرس‌وجو استفاده شده است. با استفاده از این پروفایل پرس‌وجو، دستیابی تصادفی به ماتریس جانشینی با دستیابی ترتیبی به این پروفایل جایگزین شده است. هر کدام از نخ‌های روی GPU محاسبه‌ی یک ماتریس هم‌ترازسازی را بر عهده دارند.

در [۵] روشی پیشنهاد شده که به CUDASW++ معروف است. در این پیاده‌سازی دو مرحله موازی‌سازی بین-وظیفه‌ای و درون-وظیفه‌ای در نظر گرفته شده است که در اولی، عمل هم‌ترازسازی دوگانه به یک نخ داده شده در حالی که در دومی عمل هم‌ترازسازی به یک بلاک از نخ‌ها داده می‌شود. هم‌ترازسازی توالی‌هایی از پایگاه‌داده که طول آنها کمتر از یک حدآستانه‌ی مشخص شده یا مساوی با آن باشد به مرحله اول و توالی‌هایی که طول آنها بزرگتر از حد آستانه باشد، به مرحله‌ی دوم راه می‌یابند. در روش معرفی شده در [۸]، برنامه به صورت همروند روی CPU و GPU اجرا می‌شود. هر نخ برای محاسبه‌ی یک هم‌ترازسازی دوگانه در نظر گرفته شده است. ماتریس هم‌ترازی به صورت افقی پردازش می‌شود و در هر گذر، ۱۲ سلول از نوار بالایی ماتریس پردازش خواهند شد. روش ارائه شده در [۹] از حافظه‌ی اشتراکی رو-تراشه‌ای GPU استفاده کرده، تا نقل و انتقالات داده‌ها را بین حافظه‌ی خارج از تراشه و عناصر پردازشی کاهش دهد. در این روش به جای ذخیره‌سازی کل ماتریس، فقط برای سه قطر فرعی، فضای حافظه تخصیص داده شده است. سپس این فضا برای محاسبه‌ی قطرهای فرعی بعدی مورد استفاده مجدد قرار گرفته است. در پیاده‌سازی انجام شده هر هم‌ترازسازی، به یک بلاک اختصاص داده شده و هر کدام از n نخ موجود در بلاک، مسئولیت محاسبه‌ی یک ستون ماتریس H را بر عهده دارند که هر ستون حاوی عناصر موجود روی یک قطر فرعی است.

در [۱۰] نیز پیاده‌سازی دیگری از الگوریتم اسمیت-واترمن انجام شده که پایگاه‌داده‌ی مورد بررسی در آن طی سه مرحله‌ی مرتب‌سازی، الحاق و درهم‌آمیختن به فرمت جدیدی مناسب با GPU تبدیل شده است. روش ارائه شده در [۳] فقط قابلیت هم‌ترازسازی توالی‌های طولانی DNA را بر روی GPU دارد و قادر است توالی‌های تا بیشتر از یک میلیون جفت باز را هم‌تراز کند. اخیراً بهینه‌سازی‌هایی روی این الگوریتم موازی انجام شده

است تا علاوه بر یافتن امتیاز بهینه، هم‌ترازسازی را نیز روی GPU به دست آورد [۱۱]. در پیاده‌سازی انجام شده در [۱۲] هم‌ترازسازی توالی‌های طولانی DNA انجام می‌گیرد. این روش، ماتریس هم‌ترازسازی را نصف کرده و محاسبات را به صورت جداگانه روی نیمه‌ها اجرا کرده است. برای محاسبه‌ی هر نیمه از روش به کار برده شده در [۳] استفاده کرده است. این پیاده‌سازی دو GPU را به کار برده است که هر کدام محاسبه‌ی یک نیمه را انجام داده‌اند و محاسبات با این کار سریع‌تر انجام شد. مقایسه‌ای از کارهای انجام شده در جدول ۴ ارائه شده است.

جدول ۴: مقایسه‌ی کارهای مرتبط

منبع	محاسبه مرحله سوم الگوریتم روی GPU	کارایی (GCUPS)	GPU
[۳]	ندارد	۲۰/۳	GTX 285
[۴]	ندارد	۳/۴	GTX 8800
[۵]	ندارد	۱۶/۱	GTX 295
[۶]	ندارد	۰/۶	GTX 7800
[۷]	دارد	۰/۲	GTX 7800
[۸]	ندارد	۱۴/۵	GX2 9800
[۹]	ندارد	۵/۶۵	GTX 8800
[۱۰]	ندارد	۲۱/۴	GTX 275
[۱۱]	دارد	۲۸	GTX 560

۶- نتیجه‌گیری

الگوریتم اسمیت-واترمن یکی از الگوریتم‌های محلی برای هم‌ترازسازی توالی‌های پروتئینی است که دارای دقت بالایی است. این الگوریتم دارای مراحل مقداردهی اولیه به ماتریس امتیازدهی شباهت، پر کردن عناصر ماتریس شباهت و ردیابی برای به دست آوردن بهترین هم‌ترازی است. از این مراحل، مرحله پر کردن عناصر ماتریس شباهت از نظر زمانی و هزینه محاسبات مهم‌تر است بطوریکه برای هم‌ترازسازی دو توالی A با طول ۱۰۲۴ و B با طول ۳۹۴۵، ۲۰۲ ثانیه زمان به صورت سریال صرف کرد. برای بهبود کارایی این الگوریتم این مرحله روی GPU پیاده‌سازی گردید بطوریکه هر نخ قابل پیاده‌سازی روی GPU در هر لحظه یک کاراکتر توالی اول را با چهار کاراکتر توالی دوم مقایسه کرد و بعد از اتمام این چهار کاراکتر، به همین شکل بقیه کاراکترها را محاسبه کرد و زمان آن به چهار ثانیه کاهش یافت. حداکثر کارایی در این پیاده‌سازی برای این دو توالی مختلف برابر با ۴۷ است.

مراجع

[1] D. Satyanvesh, K. Balleda, and P. K. Baruah, "GenAlign - A High Performance Implementation for Aligning the Compressed DNA Sequences," 15th International Conference on Advanced Computing Technologies, pp. 1-6, 2013.

[2] D.W. Mount, Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory Press, 2004.

[3] E.F. Sandes, and A.C. Melo, "CUDAlign: Using GPU to Accelerate the Comparison of Megabase Genomic Sequences," In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), New York, USA, vol. 45, pp. 137-146, 2010.

[4] S.A. Manavski, and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerator for Smith-Waterman sequence alignment," Journal of BMC Bioinformatics, vol. 10, 2008.

[5] Y. Liu, B. Schmidt, and D.L. Maskell, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," Journal of BMC Bioinformatics, vol. 2, 2009.

[6] W. Liu, , B. Schmidt, G. Voss, A. Schroder, and W. Muller-Wittig, "Bio-Sequence Database Scanning on a GPU," In 20th International Conference on Parallel and Distributed Processing, IPDPS, Rhodes Island, 2006.

[7] Y. Liu, W. Huang, J. Johnson, and S. Vaidya, "GPU Accelerated Smith-Waterman," International Conference on Computational Science, pp. 188-195, 2006.

[8] L. Ligowski, and W. Rudnicki, "An Efficient Implementation of Smith-Waterman Algorithm on GPU Using CUDA for Massively Parallel Scanning of sequence Databases," In IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Rome, Italy, pp. 1-8, 2009.

[9] Y. Munekawa, F. Ino, and K. Hagihara, "Design and Implementation of the Smith-Waterman Algorithm on the CUDA-Compatible GPU," In 8th International Conference on Bioinformatics and Bioengineering (BIBE), Athens, pp. 1-6, 2008.

[10] L. Hasan, M. Kentie, and Z. Al-Ars, "GPU-Accelerated Protein Sequence Alignment," In 33rdAnnual International Conference of the IEEE EMBS, Boston, Massachusetts, pp.2442-2446, 2011.

[11] E. F. O. Sandes, and A. C. M. A. Melo, "Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU," IEEE Transaction on Parallel and Distributed Systems, vol. 24, no. 4, pp. 1009-1021, 2013.

[12] M. Korpar, and M. Sikic, "SW#-GPU-enabled Exact Alignments on Genome Scale," Bioinformatics, vol. 29, pp. 2494-2495, 2013.

[13] G. M. Striemer, and A. Akoglu, "Sequence Alignment with GPU: Performance and Design Challenges," IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1-10, 2009.

[14] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology, vol. 147, no. 1, pp. 195-197, 1981.

[15] J. M. Marmolejo-Tejada, V. Trujillo-Olaya, C. P. Renteria-Mejia, and J. Velasco-Medina, "Hardware Implementation of the Smith-Waterman Algorithm using a Systolic Architecture," In 5th Latin American Symposium on Circuits and Systems (LASCAS), Santiago, USA, pp. 1-4, 2014.

[16] C. Ling, K. Benkrid, and T. Hamada, "A Parameterisable and Scalable Smith-Waterman Algorithm Implementation on CUDA-compatible GPUs," In IEEE 7thSymposium on Application Specific Processors, SASP, San Francisco, USA, pp. 94-100, 2009.

[17] NVIDIA. 2013, "NVIDIA CUDA C Programming Guide," http://docs.nvidia.com/cuda/cuda-c-programming-guide.