Iranian Association of Information and Communication Technology
IKT 2013
Shiraz University
Regular Paper

ارائه روش موازی از غربال چرخ برای تولید اعداد اول

پیمان نویدی[۱]، دکتر غلامحسین دستغیبی فرد[۲]، احسان بزرگپوری[۳]

[۱]دانشکده مهندسی برق و کامپیوتر دانشگاه شیراز ، paymanild@gmail.com
[۲]دانشکده مهندسی برق و کامپیوتر دانشگاه شیراز ، dstghaib@shirazu.ac.ir
[۳]دانشکده مهندسی برق و کامپیوتر دانشگاه شیراز ، ehsan108@shirazu.ac.ir

چکیده

این مقاله الگوریتم موازی برای تولید اعداد اول با استفاده از غربال چرخ تا یک حد مشخص را معرفی می نماید. غربال چرخ یک روش گرافیکی از غربال ارتستون است که اعداد شبه اول را از اعداد مرکب جدا می کند. امروزه با پیشرفت محاسبات موازی، امکان پردازش دستورالعمل ها بصورت همزمان توسط چندین کامپیوتر، برای کاهش زمان پردازش میسر است. ما در این مقاله الگوریتم موازی را مطرح می کنیم که با پیچیدگی زمانی بسیار پایینی اعداد اول از یک تا n را بدست می آورد. نتایج حاصل از تحقیقات ما فرمولی را بدست می آورد که احتمال تشخیص اول یا مرکب بودن یک عدد را مشخص می کند. امروزه اعداد اول نقش مهمی در رمزنگاری داشته و هنوز هم موضوعی داغ و موردعلاقه محققان است.

واژه‌های کلیدی

اعداد اول، الگوی تولید چرخ، پردازش موازی، پیچیدگی زمانی، غربال چرخ.

۱- مقدمه

ما در این مقاله می خواهیم راجع به تولید اعداد اول صحبت کنیم. اعداد اول برای مهندسان وطراحان نرم افزارهای مهندسی بسیار مهم و حیاتی است، چرا که یکی از کاربردهای اصلی اعداد اول در مسائل امنیتی و ایمنی ارتباطات رایانه ای به ویژه شبکه های مبادلاتی الکترونیک است.

غربال اعداد اول نوع سریعی از الگوریتم های یافتن اعداد اول است که غربال ارتستون یکی از الگوریتم های سریع و ساده غربال است، اما غربال پیچیده تری مانند آتکین و نوع دیگری به نام غربال چرخ وجود دارد که رایج ترند. بسیاری از این الگوریتمها برای کارایی بهتر و سرعت بالاتر از لحاظ زمانی بصورت موازی پیاده سازی شده اند. اولین الگوریتم غربال موازی در سال ۱۹۸۷ توسط آقای بخاری [۱] مطرح گردید. دلیل این کار بکارگیری غربال ارتستون برای تست در یک ماشین موازی جدید بود.

ما می خواهیم الگوریتم موازی مطح کنیم که اعداد اول بازه یک تا n را با استفاده از فاکتور چرخ بدست می آورد و این محدودیت الگوریتم های ترتیبی را به حداقل می رساند و یا به عبارتی دیگر زمان اجرا را کاهش می دهد. فاکتور چرخ کمک می کند اعداد را در چرخ های پی در پی قرار دهیم و بر اساس الگویی که به چرخ ها می دهیم اعداد مرکب را غربال کرده و اعداد شبه اول را تولید کنیم. سپس در یک فیلتر دیگر اعداد شبه اول را تمیز کرده و باقیمانده اعداد، نتیجه نهایی اعداد اول تولید شده در بازه یک تا n است.

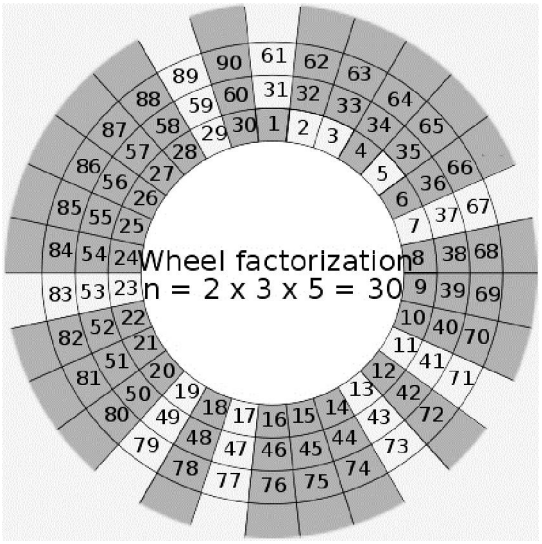
ما در این روش کار را با چند پردازنده شروع می کنیم. ابتدا پردازنده اول الگویی بر اساس مدل چرخی که می خواهیم تولید کنیم، ایجاد می کند. سپس الگو را برای همه Broadcast می کند. پردازنده های دیگر براساس الگوی دریافت شده و آدرس شروع و پایان در آرایه اصلی (که با استفاده از شماره rank هر پردازنده بدست می آید)، اعداد چرخ مربوط به خود را تولید و غربال را تبطق فرمول تعیین اعداد شبه اول انجام می دهند. در نهایت هر چرخ غربال، دارای تعدادی اعداد شبه اول است که حداکثر یک سوم طول چرخ است.

به این ترتیب، ایده اصلی ما بکارگیری روشی است که لیست بزرگی از اعداد را با پیچیدگی زمانی کمتری از روش های دیگر، غربال کند. تمام روش های غربال کارآمد براساس روشهای مشابهی کار می کنند [۲]. پیچیدگی زمانی الگوریتم غربال ارتستون O(nloglogn) است، که در الگوریتم های خطی [۴] و [۵] و [۶] به O(n/loglogn) بهبود پیدا کرده است. در نهایت کار را در قسمت دوم توضیح می دهیم و در قسمت سوم الگوریتم موازی را معرفی می کنیم. در قسمت چهارم پیچیدگی زمانی را شرح می دهیم و در قسمت پنجم نتایج حاصل از پیاده سازی و اجرای الگوریتم را مورد بررسی قراز می دهیم و نهایتاً در قسمت ششم نتیجه گیری حاصل از این تحقیق را مطرح می نماییم.

۲- شرح الگوریتم

همانطور که قبلاً توضیح دادیم می خواهیم الگوریتمی موازی را بررسی نماییم که اعداد اول بین یک تا n را با استفاده از غربال چرخ بدست می آورد. ما الگوریتم را به کمک واسط استاندارد MPICH2 بر روی سیستم عامل لینوکس اجرا نموده ایم و به نتایج مطلوبی از زمان اجرا برروی چندین پردازنده رسیده ایم.

غربال چرخ روشی گرافیکی از غربال ارتستون است که بصورت شکل ۱ اعداد را در چرخ های پشت سرهم تولید نموده و سپس با اعمال شرط لازم غربال، که در ادامه توضیح خواهیم داد، اعداد مرکب را از اعداد شبه اول فیلتر می کند. ما در ابتدا حلقه ها را با استفاده از الگوی از پیش ساخته شده می سازیم. سپس هر پردازنده حلقه ای را بصورت موازی غربال می کند. در نهایت نتایج هر پردازنده در خروجی قرار می گیرد.



شکل ۱: غربال چرخ برای اعداد ۱ تا ۹۰ با ایجاد سه حلقه ۳۰ تایی

فرض بر این است که با استفاده از MPI چند پردازنده ایجاد کنیم. پردازنده اول که با پردازنده شماره ۰ نام گذاری می شود، در ابتدا الگو را بر اساس نوع حلقه می سازد و برای پردازنده های دیگر منتشر می کند. نوع حلقه از حاصل ضرب اولین ارقام اول بدست می آید. مثلاً ۲×۳ که برابر ۶ می شود، به معنای ایجاد حلقه های شش تایی است. می توان انتخاب نوع حلقه را بر اساس تعداد اعداد n که از ورودی گرفته می شود و همچنین تعداد پردازنده هایی که می سازیم، انجام دهیم. مثلاً با چهار پردازنده و n برابر ۱۲۰ عدد، می توان نوع حلقه را ۵×۳×۲ که برابر ۳۰ می شود در نظر گرفت. انتخاب حلقه های بزرگ ممکن است کارایی الگوریتم را پایین بیاورد و ممکن است برای اعداد خیلی بزرگ تعدادی از اعداد اول بعنوان مرکب در نظر گرفته شوند و مجموع کل اعداد اول بدست آمده کمتر از حد انتظار باشد. لذا با انجام آزمایشات پی در پی به این نتیجه رسیدیم حلقه های ۳۰ تایی کمترین اشتباه را در غربال انجام می دهند. در اینصورت هر پردازنده یک حلقه دریافت می نماید. الگویی که پردازنده 0 برای پردازنده های دیگر منتشر کرد، کمک می کند تا هر پردازنده، غربال را بر روی اعداد حلقه خود اعمال کند و در نهایت اعداد شبه اول را تولید کند.

الگو با استفاده از یک الگوریتم ترتیبی توسط پردازنده 0 ایجاد می شود. پردازنده 0 اعداد اول از یک تا نوع حلقه (مثلاً ۳۰) را تولید می کند و آرایه الگو را با شروع از یک و حذف سه عدد اول ۲ ، ۳و۵ ایجاد می کند. آرایه الگو می تواند یک سوم نوع حلقه باشد. مثلاً الگویی که در مثال شکل یک ایجاد می شود شامل اعداد ۱ ، ۷ ، ۱۱ ، ۱۳ ، ۱۷ ، ۱۹ ، ۲۳ و ۲۹ می باشد. سپس الگو برای کلیه پردازنده ها Broadcast می شود. هر پردازنده به کمک رابطه (۱) و الگو دریافت شده غربال را برای تولید اعداد شبه اول انجام می دهد.

$$pattern[k] + rank \times ring\_mode \quad (1)$$

در این رابطه k از صفر شروع می شود که ابتدای آرایه الگو است و با تولید اولین عدد، به k یک واحد اضافه می شود. rank شماره پردازنده است که از صفر شروع می شود. ring_mode نوع حلقه است که در ابتدا توضیح دادیم. هر پردازنده اعدادی که در این رابطه صدق کند را به عنوان اعداد شبه اول در نظر گرفته و مابقی اعداد را حذف می کند. اعدادی که به رنگ طوسی در شکل ۱ رنگ آمیزی شده اند، اعداد مرکب اند و از حلقه حذف می شوند.

نکته دیگری که اینجا مطرح می شود این است که اگر تعداد اعداد n بسیار بیشتر از تعداد پردازنده ها باشد، در این حالت به هر پردازنده چندین حلقه می رسد. برای انجام این کار ما با تقسیم اعداد n بر تعداد پردازنده ها حلقه تکرار، که از این به بعد iteration نام می بریم، ایجاد می کنیم. هر iteration به اندازه کل پردازنده ها تکرار می شود. بفرض مثال اگر چهارپردازنده داشته باشیم، نوع حلقه ۳۰ تایی باشد و اعداد n برابر ۲۴۰ تا باشد، دو iteration خواهیم داشت. یعنی به هر پردازنده دو حلقه می رسد. اگر تعداد اعداد n دو iteration را کامل نکند، ممکن است بعضی از پردازنده ها تنها یک حلقه دریافت نمایند. برای اینکه مشخص کنیم هر پردازنده کدام حلقه ها را باید پردازش کند، طبق رابطه (۲) نقاط شروع و پایان هرکدام را در هر iteration محاسبه می کنیم.

$$start = ring\_mode \times (rank + i \times procs) + 1 \quad (2)$$
$$end = ring\_mode \times (rank + i \times procs + 1)$$

در این رابطه i شماره iteration را مشخص کرده که از صفر شروع می شود و procs تعداد کل پردازنده ها می باشد. مثلاً اگر نوع حلقه ۳۰ تایی، شماره پردازنده ۱، کل پردازنده ها برابر چهار و در iteration شماره صفر باشیم، در این صورت طبق فرمول نقطه start برابر ۳۱ و end برابر ۶۰ می شود. به این معنا که پردازنده شماره ۱ حلقه دوم از ۳۱ تا ۶۰ را در اختیار دارد.

بعد از اینکه هر پردازنده اعداد شبه اول را تولید کرد، ممکن است تعدادی عدد مرکب نیز جزو آنها باشد. در نتیجه برای اطمینان از حذف اعداد مرکب احتمالی، بر روی اعداد تولید شده که حداکثر یک سوم نوع حلقه می باشند، تابعی به نام isprime را اعمال می کنیم، تا اعداد را

غربال کند. این تابع می تواند به روش های مختلفی پیاده سازی شود. لذا برای اعداد بسیار بزرگ، می توان از سریعترین الگوریتم بررسی اعداد اول استفاده کرد، تا کارایی الگوریتم افزایش پیدا کند. ما از الگوریتم ابتدایی بررسی اعداد اول استفاده کردیم.

بعد از پایان کنترل اعداد بدست آمده، متغیر شمارنده ای را به نام f قرار می دهیم تا تعداد عدد بدست آمده هر پردازنده را حساب کند. سپس متغییر f را با استفاده از تابع Reduce از هر پردازنده جمع می کنیم و جمع نهایی را در finalsum بدست می آوریم. finalsum مجموع اعداد اول کل پردازنده ها را ذخیره می کند، اما رقم ۱ نیز جزو اعداد اول محاسبه شده و همچنین ۲، ۳ و ۵ نیز به عنوان اعداد اول محاسبه نشده اند. لذا در پایان به finalsum عدد ۲ اضافه می کنیم.

این الگوریتم کارایی بسیار بالایی برای محدوده اعداد بزرگ دارد و ما در این مقاله الگوریتم را برای دو میلیارد عدد محاسبه کرده ایم.

۳- الگوریتم موازی غربال

برای ایجاد الگوریتم غربال روالی را طی می کنیم که دستورات آن را در زبان C نوشته ایم و نیاز به تابع کمکی به نام isprime داریم، اما چون الگوریتم های مختلفی برای آن وجود دارد، لذا می توان هر الگوریتمی را جایگزین نمود و ما تشریح الگوریتم isprime را مطرح نمی کنیم.

الگوریتم در ابتدا متغییر هایی را تعریف می نماید که به شرح زیر است:

- ring_mode : که نوع حلقه را تعیین می کند. ما نوع حلقه را بصورت دستی از ورودی می خوانیم.
- max_number : حداکثر بازه برای تعیین اعداد اول را تعیین می کند. ما مقدار n برای از ورودی به این متغییر نسبت می دهیم.
- iteration : تعداد تکرار را مشخص می کند. برای ورودی n که کمتر از ضرب تعداد پردازنده ها با نوع حلقه است، این متغییر باید اصلاح شود.
- ring_counter : تعداد حلقه ها را محاسبه می کند. ممکن است تعداد حلقه ها بسیار بیشتر از تعداد پردازنده ها باشد.
- isprime : تابعی که در پایان غربال اولیه نتایج خروجی را کنترل نهایی می کند.
- pattern_array : آرایه ای که الگوی اولیه را در خود ذخیره می کند. تعداد اعداد این آرایه به وسیله متغییر pattern_counter محاسبه می گردد.
- rank و procs : به ترتیب شماره پردازنده جاری و تعداد کل پردازنده ها ها را ذخیره می کنند.
- pc : شمارنده الگو که تعداد اعداد الگو را نگه می دارد.
- f و finalsum : بترتیب جمع هر پردازنده و جمع نهایی تمام پردازنده ها را ذخیره می کنند.

در ادامه رویه الگوریتم را در تابع main مشاهده می کنید. توجه کنید که توابع MPI_Broadcast و MPI_Reduce بصورت کامل ذکر نشده، تنها متغییر های اصلی آن را عنوان کردیم:

```
int main(int argc, char **argv){
  ring_mode=atoi(argv[1]);
  max_number=atoi(argv[2]);
  ring_counter=max_number/ring_mode;
  iteration++=ring_counter/procs;
  if((max_number%ring_mode)!=0)
    ring_counter++;
  if(max_number%(ring_mode*procs)==0)
    iteration--;
  pattern_counter=ring_mode/3;
pattern_array=(int*)malloc(pattern_counter*sizeof(int))
  if(rank==0){
    k=0;
    for(j=1;j<ring_mode;j=j+2)
      if(isprime(j)=="true")
        pattern_array[++k]=j;
    pattern_array[pattern_counter-1]=k;
  }
  MPI_Broadcast(pattern_array,MPI_INT,0);
  pc= pattern_array[pattern_counter-1];
  for(i=0;i<iteration;i++){
    start=ring_mode*(rank+i*procs)+1;
    end=ring_mode*(rank+i*procs+1);
    k=0;
    for(j=start;j<=end && k<=pc;j=pattern_array[k]
                          +ring_mode(rank+i*procs)){
      if(isprime(j)=="true")
        f++;
      k++;
    }
  }
  MPI_Reduce(f,finalsum,1,MPI_INT,MPI_SUM,0);
  MPI_Finalize;
}
```

مقدار های ring_mode و max_number آرگومانهای تابع main هستند که همراه اجرای برنامه از ورودی خوانده می شوند. ring_counter از تقسیم کل اعداد ورودی بر نوع حلقه بدست می آید که تعداد حلقه های ایجاد شده را مشخص می کند. iteration تعداد تکرار را مشخص می کند که از تقسیم تعداد حلقه بر تعداد پردازنده ها بدست می آید. به iteration یک واحد اضافه کرده ایم تا اگر تعداد حلقه ها با تعداد پردازنده ها برابر یا کمتر بود، حداقل پردازنده ها یک iteration کار انجام دهند.

اگر تعداد کل اعداد بر نوع حلقه تقسم پذیر نباشد، حداقل یک حلقه اضافه ایجاد می کنیم، چون ممکن است یک سری عدد به آخر iteration

رسیده باشند و اجرا نشوند. درادامه برای ساختن الگو آرایه ای به اندازه یک سوم نوع حلقه ایجاد می کنیم.

در ابتدا پردازنده با شماره rank 0 الگو را می سازد و تعداد کل الگوی ساخته شده را در خانه آخر آرایه کپی می کند. سپس الگو به همراه تعداد کل اعداد الگو که در آرایه pattern_array قرار دارد، برای پردازنده های دیگر Broadcast می شود. دستور Broadcast پردازنده ها را تا زمان رسیدن الگو به تمام پردازنده ها همزمان می کند. بعد از دستور Broadcast بصورت همزمان تمام پردازنده ها، حتی پردازنده 0، شروع به اجرای دستورات بعدی می کنند و با افتادن در حلقه تکرار iteration، تا پایان حلقه، ring های مربوط به خود را در بازه start و end بررسی می نمایند. در هر iteration، هر پردازنده یک حلقه for را به اجرا در می آورد. حلقه for حداکثر به اندازه تعداد الگو های ساخته شده که در pc قرار گرفته اجرا می شود. متغیر j اعداد شبه اول را بدست آورده و با یک بررسی توسط تابع isprime اعداد اول نهایی بدست می آید. سپس تعداد اعداد بدست آمده در یک متغییر به نام f ذخیره می شود. در پایان حلقه تکرار، جمع اعداد اول بدست آمده از هر پردازنده با استفاده از دستور MPI_Reduce در متغییر finalsum جمع می شود. در ادامه نتایجی که از آزمایشات حاصل شده است را بررسی می نماییم.

۴- پیچیدگی الگوریتم

برای محاسبه پیچیدگی زمانی الگوریتم، دو مقدار t_{comp} زمان محاسبه و t_{comm} زمان ارتباطات بین پردازنده ها را محاسبه می کنیم. برای محاسبه t_{comm} دو مقدار باید محاسبه گردد. اول مقدار زمان Broadcast الگو برای تمام پردازنده ها، دوم مقدار زمان لازم برای تابع Reduce که مجموع بدست آمده از هر پردازنده را جمع می کند. رابطه (۳) محاسبه t_{comm} را نشان می دهد. در این رابطه r مقدار ring_counter و p تعداد پردازنده را تعیین می کند.

$$t_{Broadcast} = t_{stratup} + rt_{data} \qquad (۳)$$
$$t_{Reduce} = t_{stratup} + pt_{data}$$

$$t_{comm} = t_{Broadcast} + t_{Reduce}$$
$$= 2t_{stratup} + (r+p)t_{data}$$

همچنین برای محاسبه t_{comp} باید دو حلقه for را محاسبه کنیم. حلقه اول به اندازه ring_mode الگو را می سازد و حلقه دوم تو در تو است، حلقه بیرونی تعداد iteration را می شمارد و حلقه داخلی در هر بار برای هر پردازنده به اندازه pc تعداد الگو، حلقه ها را کنترل می کند. رابطه (۴) پیچیدگی زمانی t_{comp} را شرح می دهد. در این رابطه m تعداد ring_mode و i تعداد تکرار iteration و t تعداد الگو را تعیین می کند.

$$t_{comp} = (for_{pattern}) + (for_{iteration}) \qquad (۴)$$
$$t_{comp} = (\ m\ ) + (\ t \times i\ ) = m\ (t + i)$$
$$= n/r\ (t + i)$$
$$= i\times n/p\ (t + i\ )$$
$$= i^2 \times n/p + i\times n/p\times t$$

$$t_{comp} = O(t\times n/p)$$

مقدار i دراین رابطه تاثیر چندانی ندارد، لذا می توانیم صرف نظر کنیم. در نهایت به این نتیجه می رسیم پیچیدگی الگوریتم تابع سه مقدار ورودی است و با افزایش تعداد پردازنده ها و الگوهای کوچک تر بهترین زمان را بدست می آورد. درنتیجه پیچیدگی نهایی الگوریتم را در t_{total} بصورت رابطه (۵) بدست می آوریم.

$$t_{total} = t_{comp} + t_{comm} \qquad (۵)$$
$$= n/p\times t + 2t_{stratup} + (r+p)t_{data}$$

۵- آزمایشات و نتایج

ما برای اینکه به نتایج محسوسی برسیم، الگوریتم را بر روی سیستمی با مشخصات پایین تست کردیم. هدفمان رسیدن به کمترین زمان ممکن با کمترین سیستم موجود است. لذا آزمایشاتمان را بر روی یک دستگاه نت بوک معمولی با پردازنده Atom N570 1.6GHz شرکت Intel، با G۲ حافظه RAM تست کردیم. سپس واسط استاندارد MPICH2 را برروی سیستم عامل لینوکس اجرا نموده ایم.

در ابتدا نتایج الگوریتم یافتن اعداد اول از یک تا n را با یک پردازنده بصورت ترتیبی، بررسی نمودیم. به نتایجی از زمان که در جدول ۱ مشاهده می شود، رسیدیم. مقادیر مختلفی از n را بررسی نمودیم، در یک میلیون عدد تعداد ۷۸۴۹۸ عدد اول در زمان ۱,۳۹۸ ثانیه بدست آمد. تعداد اعداد را افزایش دادیم تا در صد میلیون عدد، تعداد ۵۷۶۱۴۵۵ عدد اول در زمان ۲۴۹۷,۸۴۸ ثانیه تولید شد. این زمان زیادی است و با افزایش تعداد ورودی، بشدت و بطور صعودی افزایش پیدا می کند. لذا الگوریتم ترتیبی برای بازه های بزرگ به هیچ عنوان کارآمد نیست.

الگوریتم را بصورت موازی با هشت پردازنده شروع می کنیم. الگوریتم نوع حلقه، تعداد n و تعداد پردازنده را از command اجرای برنامه بصورت ورودی می گیرد. تناسب خاصی برای نوع حلقه در ازای ورودی ها در نظر نمی گیریم. تنها فرض را براین اساس می گذاریم که نباید نوع حلقه کمتر از حدی باشد که پردازنده ها بسیار مشغول باشند و یا بیشتر از حدی باشد که تعداد دستورات موازی به دستورات ترتیبی نزدیک گردد. لذا انتخاب نوع حلقه و همچنین تعداد پردازنده نیز بر روی موازی سازی حائز اهمیت است و نباید تعداد پردازنده ها آنقدر زیاد باشد که باعث شود زمان ارتباط بین پردازنده و ترافیک شبکه بین آنها افزایش یابد. در الگوریتمی که ما پیشنهاد دادیم، زمان را از شروع تخصیص آرایه تا پایان عملیات Reduce محاسبه کردیم. جدول شماره ۲ نتایج بدست آمده از الگوریتم موازی پیشنهادی را برای مقادیر بسیار بزرگی از n نشان می دهد.

جدول ۱ : زمان و تعداد اعداد اول بدست آمده در الگوریتم ترتیبی

Max number	Second	Num.of prime
1.000.000	1.398	78498
2.000.000	3.674	148933
5.000.000	13.314	348513
10.000.000	35.440	664579
20.000.000	94.516	1270607
50.000.000	348.997	3001134
70.000.000	1496.980	4118064
100.000.000	2497.848	5761455

ابتدا یک میلیون عدد را با نوع حلقه ۲۳۱۰ تایی شروع می کنیم، کل اعداد در زمان ۰,۰۰۹ ثانیه معادل با نه میلی ثانیه بدست می آید. که به نسبت الگوریتم ترتیبی بسیار زمان کمی است. الگوریتم را با عدد صد میلیارد ادامه می دهیم، با نوع حلقه ۳۰۰۳۰ تایی در زمان ۰,۶۳۷ ثانیه اعداد تولید شد. که در برابر زمان تولید شده با الگوریتم ترتیبی تقریباً صفر است. در نهایت با توجه به محدودیتی که از لحاظ متغییر های حافظه در اختیارمان است، ما الگوریتم را تا عدد ۲۱۴۸۲۲۶۰۸۰ اجرا می کنیم و با نوع حلقه ۵۱۰۵۱۰ تایی به زمان ۷,۱۸۸ ثانیه خواهیم رسید. این زمان بسیار پایینی است و به نسبت الگوریتمهای موازی دیگری همچون غربال اتکین یا میلر رابین، که زمان محاسبه آنها برای دو میلیارد عدد تقریباً ۷۶ ثانیه است، بسیار مطلوب و کارآمد است.

جدول ۲ : زمان و تعداد اعداد اول بدست آمده در الگوریتم موازی

Max number	Second	Ring mode
1.000.000	0.009	2310
10.000.000	0.068	2310
50.000.000	0.189	30030
100.000.000	0.637	30030
500.000.000	1.515	30030
1.000.000.000	3.344	30030
1.500.000.000	5.197	510510
2.148.226.080	7.188	510510

۶- نتیجه گیری

این الگوریتم موازی کمک می کند تا اعداد اول بازه ۱ تا n را با سرعت بسیار بالایی تولید کنیم. الگوریتم برای داده های بزرگ بخوبی کار می کند و با افزایش تعداد پردازنده ها متناسب با اعداد ورودی و نوع حلقه، نتایج بسیار خوبی را بدست می آورد. این الگوریتم تنها الگوریتمی است که از طریق ارسال الگو و تقسیم کار برای هر پردازنده کار می کند. iteration کمک می کند بیکاری پردازنده ها به حداقل برسد. الگوریتم مشابهی در [۷] انجام شده که با استفاده از fork و بصورت درختی پیاده سازی شده است.

امید است در آینده بتوانیم با انجام آزمایشات بیشتر برای اعداد بزرگتر، به نتایج بهتری برسیم. همچنین با دسترسی به متغیر هایی از حافظه، که امکان ذخیره اعداد بزرگتری را در اختیارمان قرار دهد، صورت مسئله را بزرگتر کنیم و الگوریتم را با زمان کمتری برای اعداد بالاتر از دو میلیارد آزمایش کنیم.

۷- سپاسگزاری

با تقدیر و تشکر شایسته از استاد فرهیخته جناب آقای دکتر دستغیبی فرد که با نکته های دلاویز و گفته های بلندشان ، صحیفه های سخن را علم پرور نموده و همواره راهنما و راه گشای نگارنده در اتمام و کمال این مقاله بوده اند.

مراجع

[1] S.H. Bokhari. Multiprocessing the sieve of eratosthenes. IEEE Computer, 20(4):50–58, 1987.
[2] G. Paillard and C. Lavault. Le crible de la roue en distribu ´e. In MAJECSTIC 2003 (MAnifestation des Jeunes Chercheurs en STIC), Marseille, October 2003.
[3] C. Lavault. ´Evaluation des algorithmes distribu´es analyse, complexit´e, m´ethodes. ´Editions Herm`es, Paris, 1995.
[4] D. Gries and J. Misra. A linear sieve algorithm for finding prime numbers. Communications of the ACM, 21(12):999–1003, 1978.
[5] H.G. Mairson. Some new upper bounds on the generation of prime numbers. Communications of the ACM, 20(9):664–669, 1977.
[6] J. Sorenson. An introduction to prime numbers sieves. Technical Report 909, University ofWisconsin, Computer Sciences Department, January 1990.
[7] G. Paillard, "A Fully Distributed Prime Number Generation Using The Wheel Sieve", Universit´e Paris XIII, UMR CNRS 7030.
[8] P. Pritchard. Explaining the wheel sieve. Acta Informatica, 17:477–485, 1982.
[9] G. Paillard, C. Lavault, and F. Franc¸a. A smerbased distributed prime sieving algorithm. Technical Report 2004-04, Universit´e Paris Nord, Laboratoire d'Informatique de Paris Nord, July 2004.