Kinerja Tanda Tangan Digital RSA 1024 bit pada Simulasi E-Voting Menggunakan Prosesor Multicore

Adnan
Jurusan Teknik Elektro
Universitas Hasanuddin
Makassar, Indonesia
Email: adnan@unhas.ac.id

***Abstrak*—Paper ini menyajikan hasil pengujian kinerja program parallel algoritma tanda tangan digital RSA 1024 bit yang dimaksudkan untuk simulasi e-voting. Idenya adalah algoritma tanda tangan digital RSA diterapkan pada array bilangn *big integer* berdimensi 1024×1. Array ini mensimulasikan antrian surat suara hasil pemilihan umum. Dengan menyisipkan kode perulangan, komputasi algoritma tanda tangan digital RSA dilakukan sebanyak 600000. Perangkat lunak dikembangkan dengan melakukan paralelisasi kode perulangan tersebut. Pengujian menggunakan 24 inti prosesor menghasilkan kinerja sebesar 14286.394 tanda tangan per detik.**

***Kata Kunci*—multicore; e-voting ; work-stealing ; parallel-reduce ; kondisi-balapan;**

I. PENDAHULUAN

Hukum Moore yang menyatakan bahwa chip-chip rangkaian terintegrasi digital akan mengalami peningkatan jumlah transistor dua kali lipat setiap 18 bulan. Fenomena hukum Moore ini masih terlihat dalam perkembangan arsitektur prosesor-prosesor moderen dewasa ini. Beberapa tahun terakhir ini, arsitektur komputer-komputer cenderung berevolusi menjadi komputer parallel. Sistem multiprosesor moderen yang dikenal sebagai *multicore* sedang menjadi trend. Prosesor-prosesor *multicore* sedang mendominasi pasar. Prosesor-prosesor *multicore* dengan mudah dijumpai pada komputer-komputer laptop, dekstop, dan juga server-server. Selain itu, prosesor-prosesor *multicore* juga mengisi segment pasar cluster-cluster superkomputer.

Digunakannya prosesor-prosesor *multicore* pada kebanyakan sistem-sistem komputer berdampak pada bagaimana perangkat-perangkat lunak harus dikembangkan. Perangkat-perangkat lunak yang nantinya akan berjalan pada komputer yang berprosesor *multicore* dianggap harus mampu memanfaatkan sistem multiprosesor. Artinya, sebuah program komputer harus berjalan secara *multithreading*. Meskipun sistem operasi moderen mendukung *multithreading*, perangkat lunak harus dikembangkan menggunakan teknik pemrograman *multithreading*.

Dengan pemrograman *multithreading* pada sistem komputer berbasis prosesor *multicore* diharapkan waktu eksekusi program yang dikembangkan dapat berkurang secara signifikan. Pemrograman parallel ini sangat signifikan manfaatnya untuk mempersingkat waktu eksekusi algoritma yang memiliki kompleksitas tinggi. Waktu eksekusi dapat dikurangi secara signifikan oleh komputasi berkinerja tinggi pada sistem komputer berbasis *multicore*. Komputasi yang berkinerja tinggi dapat diperoleh jika inti-inti (cores) pada prosesor tersebut dimanfaatkan secara efisien untuk komputasi algoritma yang benar-benar diperlukan. Ini artinya, pemrograman harus dapat menekan biaya komputasi tambahan (overhead). Jadi teknik pemrograman parallel yang efisien harus diterapkan.

Teknik *multithreading* harus dapat pula mengatasi masalah *load imbalance*. Beban komputasi harus dibagi secara merata kepada seluruh thread-thread yang berjalan pada inti-inti prosesor. Jika beban komputasi tidak merata diantara prosesor, maka akan terdapat beberapa prosesor yang berbeban lebih dan ada beberapa inti yang berbeban kurang. Prosesor-prosesor yang berbeban kurang memerlukan waktu yang lebih pendek daripada inti-inti yang berbeban lebih. Pada akhirnya waktu eksekusi sebuah algoritma ditentukan oleh prosesor yang mengeksekusi tugas-tugas paling lama. Sebaliknya jika beban komputasi terbagi secara merata maka waktu eksekusi oleh semua inti-inti prosesor akan sama. Pada saat demikian, waktu eksekusi yang tercepat diperoleh.

Salah satu aplikasi yang potensial untuk memanfaatkan prosesor-prosesor multicore adalah aplikasi e-voting untuk keperluan pemilihan kepala daerah (PILKADA). Pada aplikasi e-voting seperti yang dikembangkan oleh BPPT, setiap komputer TPS tempat pemungutan suara (TPS) mengirimkan hasil perhitungan suara ke server KPU menggunakan protokol surat elektronik. Untuk keperluan otentifikasi, setiap email harus diberi *digital signature* oleh komputer TPS. Pada sisi komputer KPU, setiap email yang diterima diverifikasi keaslian pengirim dengan cara memverifikasi *digital signature* tadi. Hanya email dari komputer TPS yang asli yang akan diekstrak datanya. Kemudian komputer KPU mengakumulasi jumlah peroleh suara masing-masing calon kepala daerah. Email-email yang lain harus diabaikan.

Pada kasus aplikasi e-voting, komputer KPU harus melakukan komputasi algoritma *digital-signature* secara berulang-ulang sebanyak jumlah email yang diterimanya. Algoritma-algoritma *digital-signature* menggunakan *public-key infrastructure* seperti RSA[1] dan DSS[2] adalah algoritma yang memiliki kompleksitas-waktu yang tinggi. Jika komputer KPU memanfaatkan hanya sebuah prosesor untuk memverifikasi keaslian ribuan email dan kemudian mengakumulasi jumlah perolehan suara dari setiap komputer TPS, komputer KPU akan membutuhkan waktu yang lama untuk keperluan tersebut. Disini, prosesor-prosesor *multicore* dengan menggunakan teknik pemrograman *multithreading* dapat dimanfaatkan untuk mengurangi waktu eksekusi program e-voting.

Makalah ini merupakan pengembangan dari paper sebelumnya[3]. Pengembangan menambahkan detil formulasi algoritma perhitungan tanda tangan digital pada bagian III, dan hasil evaluasi kinerja. Adapun tujuan makalah ini yakni memperlihatkan optimisasi komputer paralel berbasis prosesor multicore dalam mengolah sejumlah data yang sangat besar dengan algoritma yang kompleks.

II. PENELITIAN TERKAIT

Cilk[4], [5] adalah perluasan bahasa C untuk pemrograman *multithreading*. Dengan perluasan, kata tersebut bermaksud bahwa Cilk menambahkan sejumlah kata kunci pemrograman C sehingga pemrogram dapat membuat program *multithreaded*. Cilk juga merupakan sistem *runtime* yang secara internal terdapat penjadwal tugas-tugas (*task scheduler*). Cilk juga memiliki sejumlah struktur data internal untuk keperluan sinkronisasi *tugas-tugas*. Penjadwal tugas Cilk menggunakan thread-thread level sisem operasi yang disebut pekerja atau *worker*. Pekerja-pekerja Cilk membuat tugas-tugas baru dengan metode *lazy-task creation*[6]. Saat pekerja membuat tugas baru (anak/child), pekerja tersebut mengalokasikan *stack-frame* baru dan menyisipkan tugas induk *task parent* ke antrian berakhir ganda miliknya (*double-ended queue*). Antrian Pekerja menyisipkan tugas-tugas pada ekor (tail) antrian. Jika pekerja kehabisan tugas baru, pekerja mengambil tugas lama dari ekor antrian miliknya sendiri. Dengan demikian pekerja menjadwalkan tugas secara *LIFO*. Jika antrian pekerja menjadi kosong, pekerja mencuri, yang kemudian disebut pencuri, tugas-tugas dari kepala ((head)) antrian milik pekerja yang lain.

StackThreads/MP adalah sistem runtime multithreading yang bekerja seperti Cilk. Berbeda dengan Cilk yang mengalokasikan stack baru dari heap, StackThread/MP menggunakan stack yang ada untuk tugas yang baru. Secara singkat, pekerja pada StackThread/MP membuat tugas baru pada dasarnya merupakan pemanggilan fungsi (*function-call*). Pada penelitian yang sebelumnya, kami mengimplementasikan konstruksi **parallel-for** berbasiskan StackThreads/MP. Konstruksi ini berbeda dengan cilk_for yang memisahkan variabel reduksi dari implementasi runtime cilk, konstruksi parallel-for mengintegrasikan variabel reduksi pada implementasi runtime. Integrasi variabel reduksi pada sistem runtime tidak memerlukan kerja kompiler yang harus dilakukan pada variable reduksi untuk cilk_for. Paper ini merupakan penerapan dari parallel-for dan variabel reduksinya.

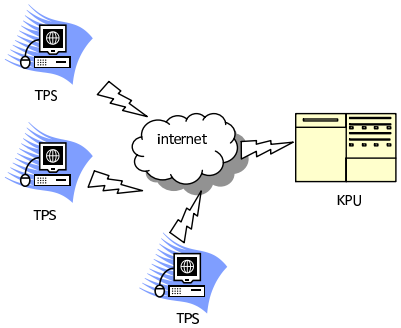
Pada makalah yang sebelumnya[3] mengusulkan metode divide-and-conquer pada bahasa cilk_for untuk melakukan paralelisasi aplikasi e-voting. Namun makalah kali ini menyajikan beberapa perbaikan serta memberikan hasil simulasi eksperimental terhadap metode tersebut.

III. SISTEM E-VOTING

Bagian ini mendiskusikan rancangan sistem e-voting dengan teknik pemrograman sekuensial dan juga dengan teknik pemrograman parallel *multithreading*. Termasuk yang didiskusikan adalah algoritma komputasi sistem e-voting.

A. *Perhitungan Suara pada sisi TPS*

Seperti yang telah di ulas pada bagian pendahuluan, sistem e-voting memungkinkan pemilih untuk memberikan suara

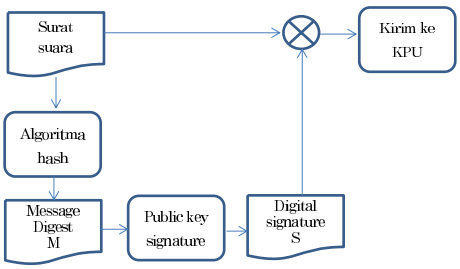


Gambar. 1: Arsitektur sistem e-voting

secara terkomputerisasi. Pada sistem e-voting, para pemilih tidak lagi memberi tanda lubang pada surat suara tetapi mereka dapat langsung menginput *vote* melalui perangkat masukan layar sentuh komputer yang berada di setiap TPS. Komputer-komputer di TPS (dua atau tiga komputer) secara terpisah menghitung perolehan suara masing-masing calon kepala daerah berdasarkan pilihan para pemilih. Pemilih hanya diberikan satu kali kesempatan memilih melalui layar sentuh komputer TPS. Secara independen, komputer TPS melakukan perhitungan perolehan suara parsial. Setelah pemilihan melewati batas waktu yang telah ditentukan, proses perhitungan suara pada komputer TPS harus berhenti dengan sendirinya. Hasil perhitungan suara parsial ditabulasi ke dalam pesan elektronik. Dengan demikian inti isi pesan elektronik adalah vektor $\bar{T}$ yang berdimensi n sama dengan jumlah calon kepala daerah. Pesan elektronik yang memuat peroleh suara masing-masing calon kemudian diberi tanda-tangan digital menggunakan kunci privat milik masing-masing TPS. Setelah pesan ditanda-tangani secara digital, pesan elektronik dikirim melalui infrastruktur internet. Gambar 1 memberikan ilustrasi konektifitas komputer TPS dan komputer KPU melalui jaringan internet.

Proses pembubuhan tanda tangan digital diperlihatkan pada gambar 2. Bukan surat suara langsung yang ditanda tangani secara digital. Yang ditanda tangani secara digital adalah *M* yaitu luaran algoritma *Hash* yang diterapkan pada surat suara tersebut. Tanda tangan digital dilakukan dengan menggunakan kunci private *d* sesuai dengan persamaan 1

$$S = M^d mod(n) \quad (1)$$



Gambar. 2: Pohon eksekusi algoritma divide and conquer parallel oleh penjadwalan workstealing

B. *Pemrosesan surat suara di KPU*

Di sini diasumsikan, pasangan kunci publik-privat telah dibangkitkan dan didistribusikan secara aman oleh KPU. Pihak KPU dapat membangkitkan pasangan kunci publik-privat dan mendistribusikan kunci privat secara aman melalui media yang aman seperti disk optik. Adapun kunci publik masing-masing komputer TPS telah disimpan pada database kunci komputer KPU.

Konsep utama dari pemrosesan surat suara pada komputer KPU adalah melakukan penjumlahan vektor suara secara terakumulasi. Dengan demikian, hasil tabulasi seluruh peroleh suara dapat dinyatakan sebagai array $T[t]$, dimana t adalah jumlah total pesan dari seluruh TPS. Sehinggal tabulasi total suara $\bar{K}$ di KPU diberikan pada persamaan 2.

$$K = \sum_{i=0}^{t-1} \bar{T}(i) \tag{2}$$

Ditunjukkan pada gambar 3, komputer KPU sebuah prosesor mengambil pesan elektronik satu-persatu. Pesan terdiri dari bagian surat suara dan bagian tanda tangan digital. Bagian surat suara diambil untuk menghitung total perolehan suara. Sedangkan bagian tanda tangan digital *S* digunakan untuk membuktikan bahwa pengirim pesan adalah sah dan juga sekaligus membuktikan isi surat suara tidak mengalami perubahan. Prosesor mengambil bagian surat suara untuk diakumulasi. Pada saat bersamaan prosesor tersebut menghitung M dari surat suara menggunakan algoritma *hash* . Prosesor kemudian mengambil kunci publik *e* yang bersesuai dengan pengirim pesan dari database. Kunci publik tersebut digunakan untuk memverifikasi keaslian pengirim pesan agar pesan yang diterima benar-benar surat suara dari TPS yang sah. Pesan dapat disimpan pada suatu bentuk struktur data *list* yang memungkinkan prosesor mengambil pesan berikutnya setelah selesai memproses pesan sebelumnya. Proses verifikasi ini dilakukan dengan menghitung M' sesuai dengan persamaan 3 yang kemudian dibandingkan dengan M. Jika M dan M' sama maka surat suara adalah asli dari TPS. Adapun algoritma pemrosesan ini ditunjukkan pada gambar 4 dan alur data diperlihatkan pada gambar 5

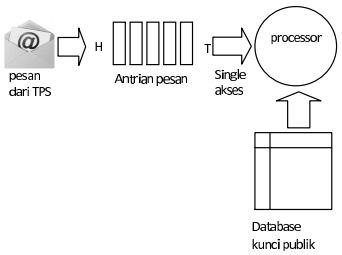
$$M' = S^e mod(n) \tag{3}$$

Disebabkan hanya ada akses tunggal oleh hanya sebuah prosesor, akses ke struktur data seperti demikian menggunakan sebuah prosesor tidak perlu menimbulkan overhead komputasi yang besar.

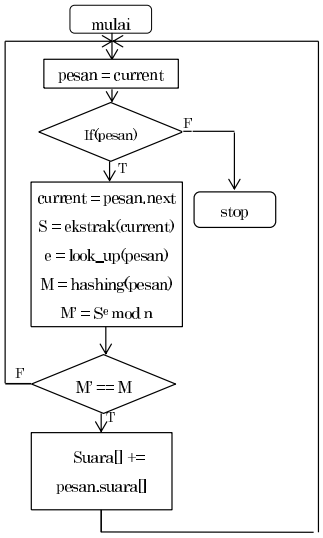
IV. Pemrograman Parallel Sistem E-Voting

Pada bagian ini didiskusikan desain sistem e-voting dengan teknik pemrograman parallel yang ditujukan untuk sistem komputer parallel berbasis prosesor multicore. Gambar 6 memperlihatkan model komputasi sistem e-voting menggunakan prosesor multicore. Teknik pemrograman parallel untuk prosesor multicore yang cocok adalah teknik *multithreading*.

Dengan teknik *multithreading*, thread-thread dapat dianggap sebagai prosesor-prosesor logika yang berbagi memori satu

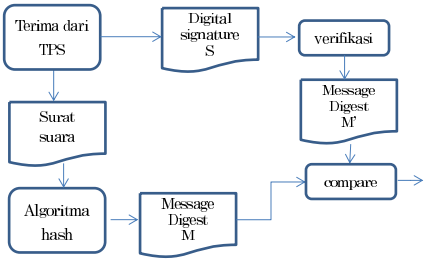


Gambar. 3: Model Antrian Pesan dan Pemrosesan Pada Komputer KPU sistem e-voting

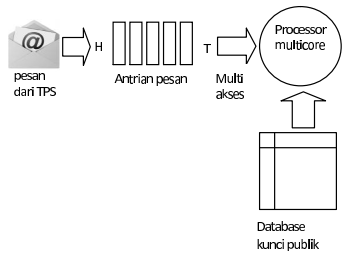


Gambar. 4: Flowchart perhitungan Suara dengan metode akumulasi vektor. Dimensi vektor sesuai dengan jumlah calon kepala daerah

sama lain. Dengan urutan yang tidak tentu, thread-thread mengakses antrian pesan. Jika antrian diimplementasikan dengan struktur senarai, thread-thread harus mengakses antrian pesan secara *mutual-exclusive*. Kabar buruk *mutual-exclusive* pada prosesor multicore yaitu berkontribusi pada overhead. Overhead tersebut akan semakin besar dengan bertambahnya jumlah inti-inti prosesor yang digunakan. Pada akhirnya eksekusi perangkat lunak tidak efisien dan mengurangi skalabilitasnya. Overhead yang besar dapat dihindari jika antrian pesan-pesan diimplementasikan menggunakan array. Index dapat diberikan pada pesan-pesan sebelum pesan-pesan diproses lebih lanjut.



Gambar. 5: Proses verifikasi keaslian surat suara



Gambar. 6: Model Antrian Pesan dan Pemrosesan Pada Komputer KPU sistem E-Voting dengan menggunakan prosesor multicore

```
N = jumlahCalon;
cilk_for(i=0;i<t;i++)
   K += pesan[i].suara
```

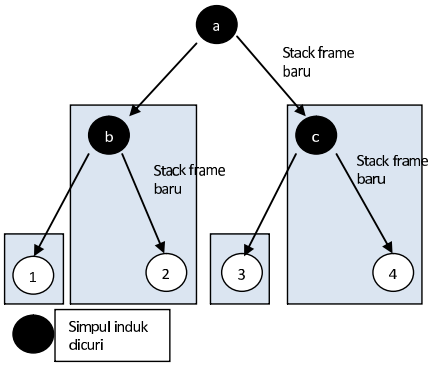
Gambar. 7: Penggunaan statemen cilk_for

A. Algoritma Divide and Conquer Parallel

Dengan adanya indeks pada antrian pesan, pemrosesan surat suara dapat dilakukan dengan teknik data parallelism, seperti menggunakan *cilk_for*. Pada sistem evoting, thread-thread dapat memproses pesan-pesan dari komputer TPS ditulis dengan statemen cilk_for seperti pada gambar 7.

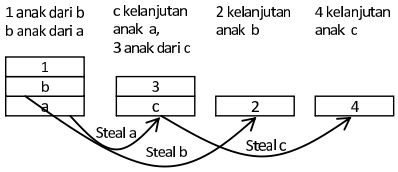
Potongan kode program pada gambar 7 sangat mungkin terjadi kesalahan mengakumulasi data suara. Kesalahan tersebut akibat kondisi balapan (race condition) terjadi pada vektor $\bar{K}$. Masalah ini dapat diatasi dengan menggunakan variabel reduksi (reduction variable).

Pada sistem runtime, cilk_for menerapkan algoritma *divide and conquer*. Algoritma ini membagi ruang iterasi $[a : b]$ menjadi dua bagian sub ruang iterasi $[a : m]$ dan $[m : b]$, dengan $m = (int)(a + b)/2$. Karena dilakukan secara rekursif, algoritma *divide-and-conquer* membentuk pohon biner seperti yang diilustrasikan pada gambar 8. Rekursi *divide-and-conquer* tidak dilanjutkan jika rata-rata parallelism (*average parallelism*) *A* sudah cukup lebih besar dari jumlah inti-inti prosesor yang tersedia.



Gambar. 8: Pohon eksekusi algoritma divide and conquer parallel oleh penjadwalan workstealing

Sub iterasi simpul-simpul terminal dijadwalkan ke thread-thread dengan teknik *work-stealing*. Pada bentuk pohon biner atau bentuk pohon seimbang yang serupa, thread-thread lebih baik mencuri simpul induk. Seperti pada ilustrasi gambar 9 sebuah thread mencuri simpul a Kemudian prosesor tersebut mengalokasikan stack-frame baru dan mengeksekusi simpul c dan kemudian 3. Sedangkan sebuah thread lain dapat mencuri simpul b lalu kemudian mengeksekusi simpul 2 dan demikian seterusnya untuk thread yang lainnya. Thread-thread mengeksekusi loop pada simpul-simpul terminal 1, 2, 3, 4 secara parallel. Iterasi-iterasi dijadwalkan pada simpul-simpul terminal tersebut. Setiap thread dapat mengerjakan loop untuk blok iterasi yang berurutan, sehingga ada thread-thread mengeksekusi badan prosedur C yang dipanggil dari simpul-simpul anak kiri. Badan prosedur C tersebut berisi loop seperti yang diperlihatkan pada gambar 10(a). Sedangkan untuk simpul anak (simpul 2 dan 4) sebelah kanan, terdapat badan prosedur C yang berisi loop seperti yang diperlihatkan pada gambar 10(b).



Gambar. 9: Ilustrasi stack-frame eksekusi thread-thread pada algoritma divide and conquer dan penjadwalan workstealing

Cara penjadwalan parallel loop yang lain yakni pada gambar 11 dengan memberikan nomor identitas id kepada node-node terminal sebagai nomor `thread_id`. Thread_id ini digunakan sebagai indeks relatif lokal untuk mengakses array pesan.

B. Variabel Reduksi pada algoritma Divide and Conquer

Kesalahan dalam mengakumulasi array $suara[][]$ secara paralel dapat terjadi akibat kondisi balapan *race condition* pada variable global vektor K seperti pada gambar 7. Kondisi balapan terhadap variable global dapat diatasi dengan menggunakan variable reduksi atau reducer[7], sehingga potongan kode yang menghilangkan race condition dengan menggunakan variabel reduction ditunjukkan pada gambar 12

Variabel reduksi adalah salinan lokal dari variabel global yang dilihat oleh sebuah thread. Thread-thread tidak langsung mengakumulasi data suara ke variabel global, tetapi ke variabel reduksi. Hasil total perhitungan suara diperoleh dengan mereduksi array $T[p]$ menjadi sebuah vektor K berdimensi n dengan operasi jumlah (`sum_reduction`) seperti yang ditunjukkan pada persamaan 4. Pada persamaan 4 r adalah jumlah banyak salinan variabel reduksi pada mana thread-thread mengakumulasi perolehan suara.

$$\bar{K} \leftarrow sum\_red(\bar{T}[r]) \tag{4}$$

Operasi reduksi pada persamaan 4 dapat dituliskan secara iteratif sebagai persamaan 5. Ide yang naif diimplementasikan untuk reduksi adalah dengan menggunakan statemen kontrol program loop yang mengakumulasi semua elemen pada index

```
for(l=a;l<m;l++){//badan loop}
```
(a) Loop untuk sub iterasi pada simpul anak kiri

```
for(r=m;r<b;r++){//badan loop}
```
(b) Loop untuk sub iterasi pada simpul anak kanan

Gambar. 10: Loop paralel yang dikerjakan oleh thread-thread dari simpul kiri dan kanan pohon biner.

```
for(i= a+thread_id; i<N; i+=NNode)
{
  //badan loop
}
```

Gambar. 11: Penjadwalan loop setiap simpul. Pada gambar, NNode adalah jumlah simpul terbawah pohon yang tidak memiliki simpul anak (terminal)

i. Namun cara ini memerluan agar semua thread-thread selesai dengan iterasi yang dikerjakan mereka sebelumnya.

$$\bar{K} = \sum_{i=0}^{p-1} \bar{T}[i] \tag{5}$$

Cara yang lebih baik yaitu mereduksi melalui pohon biner yang dibentuk oleh algoritma *divide-and-conquer*. Dengan kalimat yang lebih spesifik, reduksi dilakukan terhadap hasil perhitungan per dua simpul anak sub pohon biner. Pada gambar 13 diilustrasikan bahwa A adalah variabel global vektor suara. Variabel B, C, D, E, F, G, H adalah variabel reduksi yang dialokasikan oleh thread-thread paralel. Dengan demikian terdapat 8 thread yang mengakumulasi vektor suara $\bar{T}$ ke variabel-variabel tersebut. Pada pohon biner 8 thread saling berpasangan. Demikian pula berarti variabel-variabel yang dikerjakan mereka. Kemudian variabel reduksi simpul kanan direduksi ke variabel simpul kiri. Yakni secara formal operasi reduksi pada pohon *divide-and-conquer* dituliskan sebagai $A \leftarrow A+B$. Demikian seterusnya hingga yang diperoleh hanya sebuah simpul A yang menampung seluruh akumulasi vektor suara $\bar{T}[t]$ (baris 4).

V. EKSPERIMENT

Bagian ini menyajikan kinerja simulasi pemrosesan tanda tangan digital dalam jumlah yang sangat besar oleh program multithread berbasis komputer *multicore*. Algoritma komputasi tanda tangan digital ini adalah RSA menggunakan kunci 1024 bit. Untuk mengimplementasikan algoritma RSA menggunakan perangkat lunak fungsi pustaka GMP 4. Untuk mengimplementasikan program parallel, program ditulis dengan bahasa

```
N = jumlahCalon;
sum_reducer<VecInt[N]> K(zeroes)
cilk_for (i=0;i<t;i++)
   K += pesan[i].suara
```

Gambar. 12: Penggunaan variable reduksi dan cilk_for

1. (A) (B) (C) (D) (E) (F) (G) (H) → (A←A+B) (C←C+D) (E←E+F) (G←G+H)

2. (A) (C) (E) (G) → ((A←A+C)) ((E←E+G))

3. (A) (E) →(A←A+E)

4. (A)

Gambar. 13: Proses reduksi 8 simpul (A B C D E F G H) menjadi sebuah simpul A = (A+B+C+D+E+F+G+H) pada bohon biner divide-and-conquer.

Cilkplus dan menggunakan perangkat lunak Intel Parallel Studio. Kompilasi program dioptimasi dengan opsi -O3 -lgmp.

A. *Konfigurasi Perangkat Keras*

Penelitian ini dilaksanakan menggunakan sebuah komputer di jurusan Teknik Elektro Universitas Hasanuddin. Adapun spesifikasi perangkat keras yang digunakan adalah sebagai berikut

- 2 Prosesor AMD 6168 (total 24 cores) 1.9 MHz
- 8 GB DDR3 ECC 1066 MHz
- Mainboard Asus KGPE-D16
- Catu Daya 750 Watt

B. *Hasil*

Eksperimen dilakukan dengan menggunakan 1 hingga 24 prosesor. Setiap hasil uji coba tersebut ditampilkan waktu eksekusi dan kinerjanya seperti yang diperlihatkan pada tabel I. Data tersebut memperlihatkan bahwa dengan menggandakan jumlah prosesor, maka kinerja komputasi algoritma sangat mendekati 2 kali lipat.

TABLE I: Waktu eksekusi dan throughput 600000 tanda tangan RSA 1024 bit

Jumlah prosesor	Waktu eksekusi (milidetik)	Throughput signature/detik
1	970641	618.148
2	490672	1222.812
4	243812	2460.912
8	122709	4889.617
16	61531	9751.182
24	41998	14286.394

VI. RINGKASAN

Tujuan dari penelitian dalam makalah ini adalah untuk mengeksploitasi keunggulan sistem komputer berbasis multicore dalam mengolah sejumlah data yang sangat besar. Kasus yang dipilih adalah algoritma tandata tangan RSA dalam keperluan verifikasi keaslian simulasi surat suara dari tempat pemungutan suara yang sah. Jumlah surat suara dalam simulasi yang digunakan sebanyak 600000. Hasil eksperimen menunjukkan bahwa sistem komputer dengan 24 inti prosesor mampu melakukan 14286.39 tanda tangan digital RSA dalam 1 detik.

REFERENSI

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosytems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] National Institute of Standards and Technology, *FIPS PUB 186-2: Digital Signature Standard (DSS)*. pub-NIST:adr: National Institute for Standards and Technology, Jan. 2000.

[3] Adnan, "Metode divide and conquer parallel dan parallel-reduce pada cilk for untuk aplikasi e-voting berbasis sistem prosesor multicore," in *Proceedings of the 10th Annual on Seminar Nasional Aplikasi Teknologi Informasi (10th SNATI2013)*. Semarang, Indonesia: UII, 2013, pp. L13–L17.

[4] R. Blumofe *et al.*, "Cilk: An efficient multithreaded runtime system," *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 55–69, 1996.

[5] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the Cilk-5 multithreaded language," *ACM SIGPLAN Notices*, vol. 33, no. 5, pp. 212–223, May 1998.

[6] E. Mohr, D. A. Kranz, and R. H. Halstead, Jr., "Lazy task creation: A technique for increasing the granularity of parallel programs," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PDS-2, no. 3, pp. 264–280, Jul. 1991.

[7] M. Frigo, P. Halpern, C. E. Leiserson, and S. Lewin-Berlin, "Reducers and other cilk++ hyperobjects," in *Proceedings of the 21st Annual ACM Symposium on Parallel Algorithms and Architectures (21st SPAA'09)*. Calgary, Alberta, Canada: ACM SIGACT & ACM SIGARCH, Aug. 2009, pp. 79–90, cilk Arts.