A Reconfigurable Sparse Matrix-Vector

Multiplication Engine on FPGAs

Josses

+628-998-668-963

josses_pmr@yahoo.com

Sparse matrix-vector multiplication (SMVM) adalah dasar yang digunakan untuk banyak aplikasi high-performance computing, seperti penerimaan informasi, image construction, medical imaging, industrial engineering, control sistem simulation, dan economic modeling [1]. Di sisi lain, teknologi reconfigurable computing dapat digunakan untuk melakukan perhitungan dengan fleksibilitas yang tinggi seperti GPPs(General Purpose Processors) dan dengan performance yang tinggi seperti ASICs(Application Specific Integrated Circuits) [2]. Aplikasi SMVM dengan rangkaian hardware biasa akan mengakibatkan turunnya kinerja keseluruhan sistem karena adanya beberapa pipeline stalls selama komputasi (karena adanya zero padding) atau jumlah input yang melebihi batas (preprocessing). Untuk skala sparse matrix yang lebih besar, ini akan menimbulkan overhead dari performance yang sangat tinggi dan menurunkan kemampuan FPGA secara drastis. Paper ini akan membahas design FPGA yang efisien berbasiskan arsitektur SMVM yang dapat menghandle matrix tanpa preprocessing dan zero padding yang besar dan dapat diexpand secara dynamic [1] dengan memanfaatkan teknologi reconfigurable computing [2] sesuai dengan orde multiplier yang dibutuhkan oleh programmer dengan memanfaatkan instruction set seperti pada GPPs yang fleksibel.

Kata kunci: *FPGA, reconfigurable computing, sparse matrix-vector multiplication*

PENDAHULUAN

Sparse matrix-vector multiplication (SMVM) memegang peranan penting untuk banyak aplikasi scientific dan engineering, seperti image construction, medical imaging, industrial engineering, control sistem simulation, dan economic modeling. Untuk menyelesaikan linear sistem yang besar, y=Ax (dimana A adalah *n* x *n* sparse matrix dengan nilai n_z bukan nol, dan x dan y adalah matrix *n* x 1) dan dengan eigenvalue (λ) Ax= λx menggunakan metode berulang, SMVM dapat dieksekusi sebanyak ratusan atau ribuan kali dengan matrix yang sama. Sebagai contoh, masalah PageRank Google eigenvector yang dihandle dengan SMVM dimana ukuran matrixnya (*n*) bisa mencapai milyaran. Bila ini dikerjakan secara sekuensial, maka penyelesaiannya akan menghabiskan waktu yang sangat lama [3]. Tujuan dari tulisan ini adalah untuk meminimalisir memory overhead tanpa preprocessing input matrix yang terlalu banyak, seperti yang telah dilakukan dengan

pendekatan sebelumnya [4]. Design yang akan dibangun juga akan mengeliminasi penggunaan zero padding (stalls), namun akan menimbulkan sedikit overhead dari hardware tambahan bila dibandingkan dengan pendekatan sebelumnya.

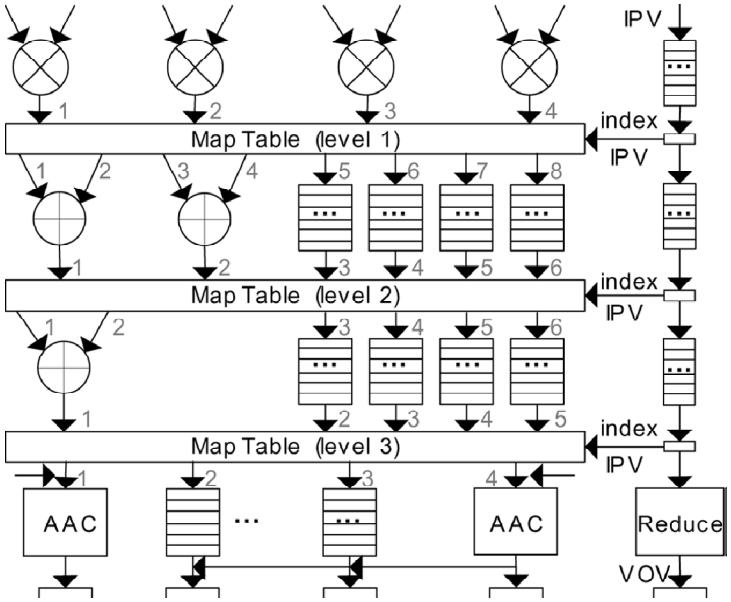
Kemampuan Reconfigurable Computing yang dimiliki FPGA berada diantara GPPs (tidak efisien namun fleksibel) dan ASICs (efisien namun tidak fleksibel). Kemampuan yang dimiliki FPGA ini menggabungkan efisiensi ASICs yang merupakan rangkaian yang hardwired dan GPPs yang dapat diatur oleh programmer secara fleksibel dengan instruction set yang telah tersedia.

Karena pentingnya SMVM dalam keperluan scientific dan engineering, banyak yang telah mencoba untuk memaksimalkan kinerja dari SMVM. Penulis di [5] menjelaskan teknik pada level instruction parallelism pada processor superscalar RISC. Penulis di [6] melakukan research untuk mendistribusikan elemen (n_z) non-zero dengan parallel processor array. Design yang dibuat di [7] menggunakan pipeline stalls yang jumlahnya tetap, tidak tergantung dari struktur matrix. Design yang akan dibahas di sini akan menggabungkan hasil dari beberapa penelitian yang telah dilakukan sebelumnya, khususnya penelitian pada [1] dan [2].

METODE PENELITIAN

Teknik penyimpanan dan pemrosesan nilai nonzero dalam sparse matrix tetap dipergunakan, namun ditambah dengan kemampuan mengurangi jumlah memory yang terpakai dan meningkatkan performance. Kemudian, setelah design ini selesai dibangun, penambahan fitur reconfigurable architecture untuk design ini ditambahkan sehingga design multiplication engine ini menjadi direconfigureable dengan jumlah multiplier yang diinginkan oleh programmer. Arsitektur yang akan dibuat mengasumsikan bahwa sparse matrix telah diurutkan per baris dengan nilai nonzero. Penambahan multiplier array (k multiplier) dan sebuah binary adder tree. Setiap clock cycle, multiplier bekerja mengalikan elemen nonzero yang ada di baris yang sama dalam sparse matrix. Jika jumlah nonzero di tiap baris l_i sama dengan k, maka arsitektur ini efisien, jika lebih kecil, maka akan terdapat zero padding, dan jika lebih besar, maka terdapat pipeline stalls. Tujuan akhir dari design ini

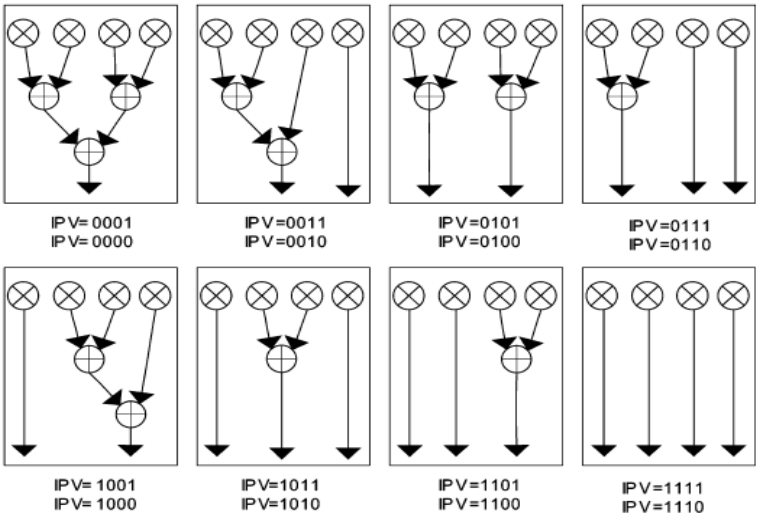
adalah menghilangkan kerugian ini hanya dengan menambahkan arsitektur yang tidak terlalu rumit.



Gambar 1. Arsitektur SMVM (*k*=4)

3.1 Input Pattern

Gambar 1 adalah rangkaian SMVM dengan 4 multiplier (*k*=4). Setiap elemen matrix ditransfer ke sistem ini dalam 1 clock cycle tanpa mempedulikan apakah elemen tersebut dari baris yang sama atau tidak. Nonzero dari keseluruhan sparse matrix dapat dibagi menjadi banyak segment tergantung dari banyaknya *k*. Setiap segment terdiri dari *k* nilai yang akan dimasukkan ke tiap multiplier yang nantinya akan dikalikan dengan koefisien masing-masing. Input Pattern Vector (IPV) untuk tiap segment didapatkan dari tiap bit dari *k*-bit. Bit ke-*i* dari *k*-bit IPV sama dengan 1 bila nilai ke-*i* di segment tersebut adalah nilai nonzero terakhir dari suatu baris, selain itu, nilainya adalah 0. Nilai IPV ini dapat memberikan informasi baris sehingga penjumlahan akhir dari suatu baris dapat dilakukan karena jumlah multiplier dan adder yang tidak sama dengan jumlah nilai nonzero. Contoh dapat dilihat di gambar 2. Nilai dari IPV akan menentukan apakah nilai dari elemen yang bersangkutan akan dijumlahkan dengan cycle clock yang sama atau akan dijumlahkan dengan nilai yang didapat di clock cycle selanjutnya atau sebelumnya.



Gambar 2. Input pattern dan tree structure

3.2 Arsitektur SMVM

Aristektur yang akan dibuat di Gambar 1 terdiri dari multiplier array, adder tree, 2 adder accumulator (ACC), map table, dan register array. Arsitektur ini adalah arsitektur pipeline dan bekerja secara parallel. Data mengalir dari atas ke bawah seperti bentuk pipeline biasa. Setiap clock cycle, *k* nilai dari matrix akan dikalikan dengan dengan *k* nilai dari vector. Banyaknya multiplier (*k*) dapat di-rekonfigurasikan dengan bilangan bulat lainnya. Koneksi antara output dari multiplier, input dari adder, register array dan AAC ditentukan dari look up table dari masing-masing map table. Map table akan mengkonfigurasikan tergantung dari nilai IPV yang ada dari masing-masing segment.

3.3 Multiplier dan Adder

Multiplier yang dipergunakan adalah multiplier pipelined yang akan menampilkan hasil output perhitungan, beberapa clock setelah input diberikan. Nilai input diberikan tiap clock cycle. Kemudian, berdasarkan IPV, beberapa hasil perkalian (product) dijumlah bila nilai tersebut masih berada dalam 1 baris matrix. Ada juga hasil product yang tidak perlu dimasukkan ke adder tree, melainkan hanya diteruskan ke register array (lihat Gambar 2) dan masuk ke level pipeline berikutnya bersamaan dengan nilai dari adder tree pada clock cycle tersebut. Adder tree memiliki $^{2}\log k$ level. Register array adalah 64-bit FIFO queue. Jumlah register dari array tiap level sebanyak *k* untuk memastikan semua product dapat masuk bila semua bit nilai IPV = 1.

3.4 Adder Accumulator (AAC)

Arsitektur ini menggunakan 2 buah AAC untuk menjumlahkan hasil akhir dari tiap baris. Cara kerja AAC ini adalah seperti berikut : AAC pertama akan bekerja terus menjumlahkan dari tiap segment hingga terdapat informasi *end of row* dari IPV. Ketika informasi tersebut didapat, maka hasil akhir dari penjumlahan nilai tersebut disimpan ke dalam register array penampung yang akan merupakan output dari baris yang bersangkutan. Selama AAC pertama bekerja untuk menjumlahkan hasil akhir dan menyimpan ke dalam register, AAC kedua berperan untuk menjumlahkan nilai dari baris berikutnya yang datang bersamaan dengan signal *end of row* dari baris sebelumnya. AAC akan mengeluarkan output hasil 2 clock cycle setelah nilai input akhir diterima. Output sementara dari AAC ini disimpan ke salah satu dari *k*-2 register array yang tersedia. Register array dengan jumlah *k*-2 ini ditujukan untuk dapat memastikan agar tiap nilai sementara maupun nilai akhir dari AAC dapat ditampung.

3.5 IPV Reduction

Jumlah dari IPV dapat direduksi tiap kali terjadi penggunaan adder tree. Penggunaan adder tree mengakibatkan berkurangnya total nilai yang ada di tiap level pipeline, dengan demikian, IPV reduction dapat dilakukan. Pattern dari sepasang IPV yang dapat direduksi adalah (0,0) atau (0,1) yang akan direduksi menjadi 0 atau 1 dan dikirim ke level selanjutnya. Bit-bit yang tidak termasuk ke dalam pattern tersebut akan dikirimkan langsung ke level selanjutnya. Prosedur IPV reduction ini berlangsung selama $^{2}\log k$ kali. Reduced IPV (RIPV) memiliki jumlah bit berlogic 1 yang sama dengan IPV original dengan semua bit set “didorong” ke kiri (Figure 4 hal 116 paper ke-[1]).

3.6 Map Table

Map table adalah array 2D dan memberikan koneksi antara input dan output pada masing-masing level pipeline. Indeks dari map table ini ditentukan dari nilai integer IPV. Input dan output dari tiap level diberikan nomor dari 1. Tabel 1 menunjukkan map table jika *k*=4. Level pertama adalah mapping antara multiplier dengan register dan adder tree. Level kedua adalah mapping antara register dan adder tree. Level ketiga adalah mapping antara register dan adder tree dengan register dan AAC.

		Level 1								Level 2						Level 3			
IPV	**Index**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**
000X	**0**	1	2	3	4	0	0	0	0	1	2	0	0	0	0	1	0	0	0
001X	**1**	1	2	0	0	3	4	0	0	1	3	4	0	0	0	1	0	0	2
010X	**2**	1	2	3	4	0	0	0	0	0	0	1	2	0	0	2	0	0	3
011X	**3**	1	2	0	0	3	4	0	0	0	0	1	3	4	0	2	3	0	4
100X	**4**	2	3	0	0	1	4	0	0	1	4	3	0	0	0	2	0	0	1
101X	**5**	2	3	0	0	1	4	0	0	0	0	3	1	4	0	2	3	0	4
110X	**6**	3	4	0	0	1	2	0	0	0	0	3	4	1	0	2	3	0	4
111X	**7**	0	0	0	0	1	2	3	4	0	0	3	4	5	6	2	3	4	5

Tabel 1. Map table ketika k=4

3.7 Output

Tidak semua k-2 register array yang terdapat pada level terakhir adalah nilai yang valid. Karena itu, kita membutuhkan k-bit Valid Output Vector (VOV) untuk mengindikasikan bahwa output yang tersedia pada tiap clock cycle adalah nilai yang valid. Jika bit ke-i dari VOV adalah 1, maka output register ke-i merupakan output yang valid, sebaliknya, bila tidak, maka outputnya masih merupakan sebagian dari jumlah suatu baris. Kita dapat melihat hubungan antara IPV dan VOV dengan jelas. Berdasarkan fungsi dari IPV, IPV akan bernilai 1 jika nilainya adalah akhir dari suatu baris. Karena itu, nilai VOV bisa didapatkan dari informasi yang didapat dari IPV.

HASIL DAN PEMBAHASAN

Performance dari keseluruhan sistem ini ditentukan oleh kecepatan dari SMVM. SMVM adalah tipikal aplikasi data-intensive dimana kecepatan berbanding lurus dengan banyaknya data. I/O bandwidth juga mempengaruhi performance dari data-intensive computing, namun I/O bandwidth dan jumlah pin yang dibutuhkan dari chip FPGA tergantung dari strategi design dan dimana data asli disimpan. I/O bandwidth dalam sistem ini dapat diabaikan karena data disimpan di dalam on-chip memory [1]. Jumlah pin juga dapat dikurangi secara signifikan bila modul SMVM dibungkus dengan modul lain. Sementara itu, loading time tidak dapat diabaikan karena data yang akan dihitung disimpan dalam storage external (hard disk, network storage, dll).

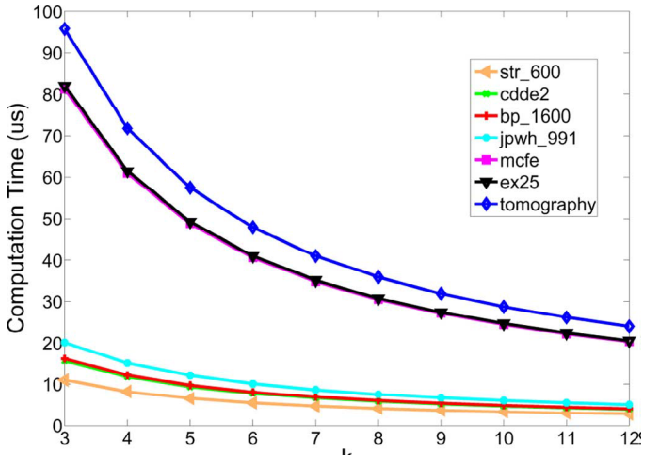
Arsitektur SMVM yang dibahas pada paper ini dapat dianggap sebagai sebuah sistem dengan k Processing Elements (PE). Setiap PE memproses 1 element dari input matrix dalam 1 clock cycle. Bila keseluruhan matrix dibagi menjadi n_z/k blok, maka semua element di 1

blok dapat diproses dalam 1 clock cycle dengan blok yang dimasukkan ke sistem secara sekuensial. Metode ini disebut juga *locally parallel globally sequential (LPGS)* pada [9]. Jika semua sistem SMVM dianggap sebagai 1 PE dan menggunakan beberapa FPGA, maka dapat disebut *locally sequential globally parallel (LSGP)* [9]. Pekerjaan selanjutnya adalah mengimplementasikan arsitektur ini ke platform Convey HC-1 dengan metode LSGP. Jumlah Processing Element k adalah faktor yang reconfigurable yang dapat diatur oleh programmer menggunakan instruction set yang telah tersedia oleh CPU.

a. Experimental Setup : XD2000i board dari XtremeData terdiri dari Dual Xeon motherboard dengan 1 Intel Xeon processor dan 2 Altera Stratix III EP3SE260 FPGA. CPU Xeon memiliki 4Gb sistem memory dan 4core yang running di 1,6GHz yang berkomunikasi dengan FPGA melalui Intel Memory Controller Hub (MCH). MCH terkoneksi dengan FPGA melalui interface 1067 M Front Side Bus (FSB). FSB ini dapat bekerja maksimal 8,5GBytes/s communication bandwidth yang merupakan performa tertinggi dan latency bus yang terendah dari platform Intel [14]. Clock FPGA tersambung ke 100MHz secara hardwired. Software yang digunakan untuk simulasi, sintesis, place dan route adalah Altera Quartus II 8.1. Software lain yang digunakan sebagai perbandingan adalah SparseLib++ yang merupakan general-purpose software implementation dengan bahasa object oriented C++ dengan sparse matrix library [15][16][17].
b. Parameter Design : Nilai dari *k* yang merupakan banyaknya multiplier disetting dari 3 hingga 12. Kompleksitas design melebihi batas dari Quartus FPGA synthesis tools untuk *k*>12. Arsitektur SMVM menggunakan 62% dari total logic di Stratix III EP3SE260 FPGA ketika k=12. Komponen pada keseluruhan sistem XD2000i habis dipakai sebanyak 14% dari total LUT resource yang tersedia menggunakan arsitektur SMVM.
c. Experimental Results : Untuk mendapatkan hasil yang valid, maka SparseLib++ juga akan berjalan diatas platform XtremeData 2000i menggunakan 1 CPU Intel Xeon. Pengambilan data dilakukan dengan beberapa matrix yang berbeda dengan nilai nonzero yang berbeda juga. Tabel 3 menunjukkan karakteristik dari sparse matrix yang digunakan yang didapat dari University of Florida Sparse Matrix Collection [18].

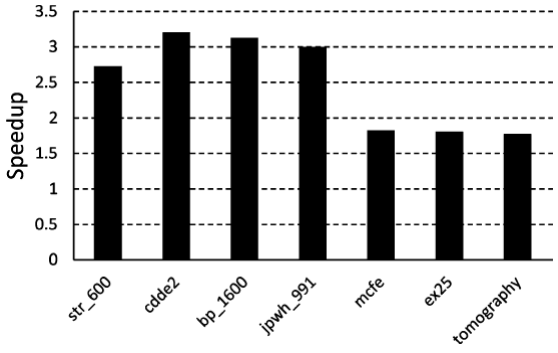
Name	Size	Nonzeros	t_{read}	t_{trans}	t_{ipv}
str_600	363×363	3279	6459	61	36
cdde2	961×961	4681	10741	87	44
bp_1600	822×822	4841	9387	90	63
jpwh_991	991×991	6027	9660	112	78
mcfe	765×765	24382	41630	452	171
ex25	848×848	24612	54897	457	173
tomography	500×500	28726	60897	533	200

Tabel 3. Karakteristik SMVM matrix



Gambar 3. Komputasi FPGA, waktu komputasi berdasarkan fungsi dari *k*

Gambar 3 menjelaskan tentang waktu komputasi terhadap banyaknya *k* yang dipergunakan. Gambar 4 menjelaskan tentang speedup FPGA terhadap CPU (rasio running time t_{FPGA}/t_{CPU}). Perlu diketahui bahwa design ini mampu bekerja pada frekuensi clock yang lebih tinggi menggunakan Stratix III EP3SE260 FPGA yang akan mengurangi waktu komputasi secara linear.

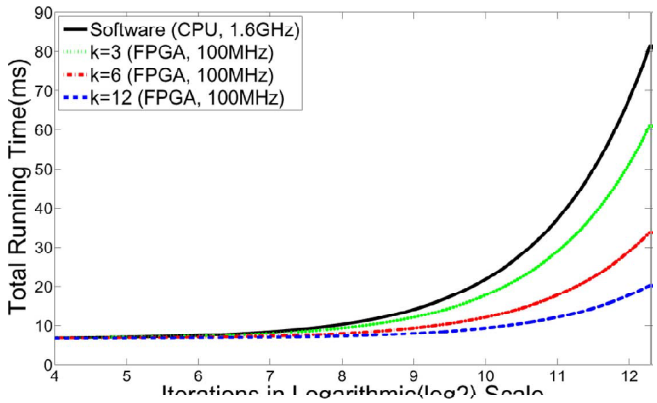


Gambar 4. Speedup FPGA terhadap CPU untuk *k*=6

t_{read} pada tabel 3 adalah waktu untuk membaca matrix dan vector dari disk ke sistem memory. Besarnya file ditentukan oleh banyaknya digit mantissa, karena itu t_{read} untuk ccde2 lebih besar daripada bp_1600 walaupun ccde2

memiliki nonzero yang lebih sedikit. t_{trans} adalah waktu yang dibutuhkan untuk mentransfer data dari sistem memory ke chip FPGA. t_{IPV} adalah waktu yang dibutuhkan oleh software untuk menggenerate IPV. Semua waktu ini dalam satuan mikrosekon. Total waktu untuk preprocessing adalah t_{read} + t_{trans} + t_{IPV}.

Finite State Machine (FSM) digunakan untuk mengatasi defisiensi dari input SMVM engine. FSM bertugas untuk mengambil data nonzero dan IPV, kemudian memberikannya ke rangkaian SMVM engine. FSM diimplementasikan pada rangkaian FPGA dengan VHDL code. Jika buffer pada FSM sudah penuh (jumlah data nonzero sama dengan *k*), maka data tersebut akan diberikan ke SMVM engine. Sebaliknya, bila masih belum penuh, maka FSM akan terus mengambil data. Untuk m=1 pada $Y=A^mX$, dimana m adalah jumlah pengulangan SMVM, waktu komputasi jauh lebih kecil dibandingkan dengan waktu preprocessing. Untuk nilai m yang kecil, sistem CPU bekerja lebih cepat daripada FPGA karena adanya preprocessing overhead. Namun pada prakteknya, nilai m besar sehingga SMVM banyak dipakai untuk melakukan komputasi. Perbandingan antara CPU dan FGPA dapat dilihat pada Gambar 5.



Gambar 5. Perbandingan running time untuk data str_600

Harus diketahui bahwa input matrix disimpan secara kontinu on-chip. Yang menjadi permasalahan adalah memory on-chip yang tersedia. Pada chip Altera Stratix III EP3SE260 terdapat on-chip memory M144K dengan besar 64 x 4k (width x depth) yang digunakan untuk menyimpan matrix dan vector.

Ketika *k*=6, 40 blok M144K digunakan sebagai I/O storage. Karena totalnya hanya ada 48 blok M144K, maka jumlah nonzero pada matrix tidak bisa lebih dari 4k [19].

KESIMPULAN

Paper ini membahas tentang reconfigurable component dari arsitektur high performance sparse matrix-vector multiplication. Aliran data yang masuk ke arsitektur ini ditentukan oleh map table secara dinamis pada saat runtime. Arsitektur ini dapat digunakan untuk sparse matrix dengan ukuran apapun. Arsitektur ini juga mengeliminasi zero padding dan pipeline stalls dengan menggunakan IPV. Dengan bandwidth komunikasi dan resources hardware yang cukup, performance dari arsitektur ini meningkat secara linear bersamaan dengan meningkatnya jumlah multiplier (*k*) yang merupakan parameter yang reconfigureable. Implementasi arsitektur ini dilakukan pada board XtremeData2000i reconfigurable platform dari Altera. Bila dibandingkan dengan optimasi software, design ini jauh lebih cepat untuk komputasi. Arsitektur SMVM yang diimplementasikan pada platform XD2000i lebih unggul daripada pendekatan tradisional ketika sumber data tersimpan di dalam on-chip memory FPGA atau rangkaian SMVM digunakan berulang kali ($Y=A^mX$, untuk $m >> 1$). Arsitektur ini tidak terbatas pada platforrm ini.

Kedepannya, arsitektur SMVM dapat diimplementasikan pada platform baru, Convey HC-1[8] yang memiliki kelebihan yaitu adanya multiple FPGA dan independent memory controllers. Teknik mempartisi matrix yang besar untuk komputasi berulang pada FPGA merupakan aspek yang juga penting untuk dikembangkan ke depannya karena adanya keterbatasan on-chip memory dari FPGA.

DAFTAR PUSTAKA

[1] Song Sun, Madhu Monga, Philip H. Jones, Joseph Zambreno, "An I/O Bandwidth-Sensitive Sparse Matrix-Vector Multiplication Engine on FPGAs," *IEEE TRANSACTIONS ON CIRCUITS AND SISTEMS*, vol. 59, no. 1, pp. 113-117, January 2012

[2] M. Aqeel Iqbal, Uzma Saeed Awan, Shoab a. Khan, "Reconfigurable Computing Technology Used for Modern Scientific Applications," *IEEE International Conference on* Education Technology and Computer (ICETC), vol. 5, no. 2, pp. 36-41, 2010

[3] A. N. Langville and C. D. Meyer, Eds., Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton, NJ: Princeton Univ. Press, 2006.

[4] K. Underwood, K. S. Hemmert, and C. Ulmer, "Architectures and APIs: Assessing requirements for delivering FPGA performance to applications," presented at the Supercomputing (SC) 2006.

[5] S. Toledo, "Improving memory-sistem performance of sparse matrixvector multiplication," IBM J. Res. Develop., 1997.

[6] A. T. Ogielski and W. Aiello, "Sparse matrix computations on parallel processor arrays," SIAM J. Sci. Comput., vol. 14, pp. 519–530, 1993.

[7] J. Sun, G. Peterson, and O. Storaasli, "Sparse matrix-vector multiplication design on FPGAs," presented at the Int. Symp. Field-Programmable Custom Computing Machines, Apr. 2007.

[8] Convey Computer Corporation [Online]. Available: http://www.conveycomputer.com

[9] S. Y. Kung, Ed., VLSI Array Processors. Upper Saddle River, NJ:Prentice-Hall, 1988.

[10] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," presented at the Int. Symp. Field Programmable Gate Arrays (FPGA), Feb. 2005.

[11] Y. El-Kurdi, D. Giannacopoulos, and W. J. Gross, "Hardware acceleration for finite-element electromagnetics: Efficient sparse matrix floating-point computations with FPGAs," IEEE Trans. Magn., vol. 43, no. 4, pp. 1525–1528, Apr. 2007.

[12] M. deLorimier and A. Dehon, "Floating-point sparse matrix-vector multiply for FPGAs," presented at the Int. Symp. Field Programmable Gate Arrays (FPGA), Feb. 2005.

[13] Y. Zhang, Y. Shalabi, R. Jain, K. Nagar, and J. Bakos, "FPGA vs. GPU for sparse matrix vector multiply," in Proc. Int. Conf. Field-Programmable Technology (FPT), 2009, pp. 255–262.

[14] L. Ling, N. Oliver, C. Bhushan, W. Qigang, A. Chen, S.Wenbo, Y. Zhihong, A. Sheiman, I. McCallum, J. Grecco, H. Mitchel, L. Dong, and P. Gupta, "High-performance, energy-efficient platforms using in-socket FPGA accelerators," in Proc. Int. Symp. Field Programmable Gate Arrays (FPGA), 2009, pp. 261–264.

[15] R. Pozo, K. Remington, and A. Lumsdaine, Sparselib++ [Online]. Available: math.nist.gov/sparselib++/

[16] J. Dongarra, A. Lumsdaine, X. Niu, R. Pozo, and K. Remington, Sparse Matrix Libraries in C++ for High Performance Architectures, 1994.

[17] I. S. Duff, M. A. Heroux, and R. Pozo, The Sparse BLAS CERFACS, Technical Report TR/PA/01/24, 2001.

[18] T. Davis and Y. Hu, The University of Florida Sparse Matrix Collection [Online]. Available: www.cise.ufl.edu/research/sparse/matrices

[19] Altera, FPGA Coprocessing Evolution : Sustained Performance Approaches Peak Performance, White Paper, dated June 2009, ver 1.1.