Massively Parallel Processor based on Systolic Architecture for High-Performance Computation of Difference Schemes

Kentaro Sano*, Takanori Iizuka and Satoru Yamamoto

Department of Computer and Mathematical Sciences
Graduate School of Information Sciences, Tohoku University
6-6-01 Aramaki Aza Aoba, Sendai, 980-8579, Japan
E-mail:kentah@caero.mech.tohoku.ac.jp - Web page: http://www.caero.mech.tohoku.ac.jp

Key Words: Systolic Architecture, Difference Scheme

ABSTRACT

While the computational requirements for computational fluid dynamics (CFD) have been increased, today's supercomputers are confronted with difficulties in performance improvement at reasonable cost. This is because of their general-purpose feature resulting in inefficient hardware-resource utilization to accelerate any applications without discrimination. Such circumstances have motivated researchers to develop special-purpose processors tailored for each individual application. In this paper, we propose a massively parallel processor for high-performance difference scheme computations, which is based on the systolic architecture. We show an FPGA-based prototype design to accelerate the fractional step method [1], and discuss its expected performance.

Several researchers have studied special-purpose computers for CFD. Hauser showed initial efforts of an FPGA-based computer for a basic, compressible flow solver[2]. After describing a numerical method of the flow solver, he reported that operations of vectors and matrices; dot-product, matrix-vector and matrix-matrix multiplication, were implemented to be used as basic components of the flow solver. Although the attempt to utilize FPGAs for a CFD solver is interesting, the algorithm and its parallelism to be exploited were not discussed well from aspects of computer architectures.

We focus on the systolic architecture for a systolic algorithm to efficiently perform difference schemes. An algorithm that can efficiently be performed on a systolic array is referred to as *a systolic algorithm*. The systolic array proposed by H.T. Kung et al. [3] is a regular arrangement of many simple processing elements (PEs) in an array where data are processed and flow synchronously across the array between neighbours. In addition to massive parallelism of the systolic array, our proposed systolic computational memory architecture [4][5] has an advantage of wide and scalable memory-bandwidth due to local memories distributed among PEs. Thus the systolic array has significant scalability in terms of computation and memory access, which avoid the bottleneck of the conventional general-purpose microprocessors. In a basic systolic algorithm, at each step, individual PE takes in data from one or more neighbours, processes them and outputs the results to other neighbours. A difference scheme that gives discrete forms of differential equations inherently matches the systolic algorithm because it relies on calculation with values of adjacent lattice points.

While the general curvilinear coordinate can be applied to our proposal, we use a 2D regular orthogonal grid as an example for brief discussion. A incompressible flow governed by the following equations is solved by the fractional step method.

$$\nabla V = 0, \tag{1}$$

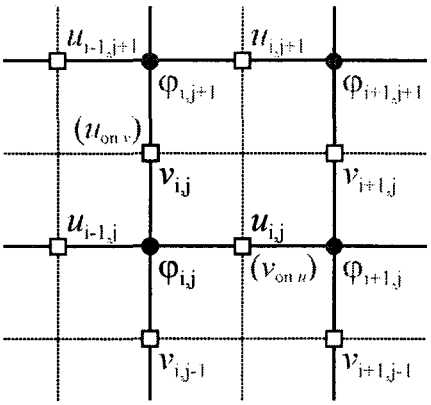

Fig. 1 2D staggered mesh.

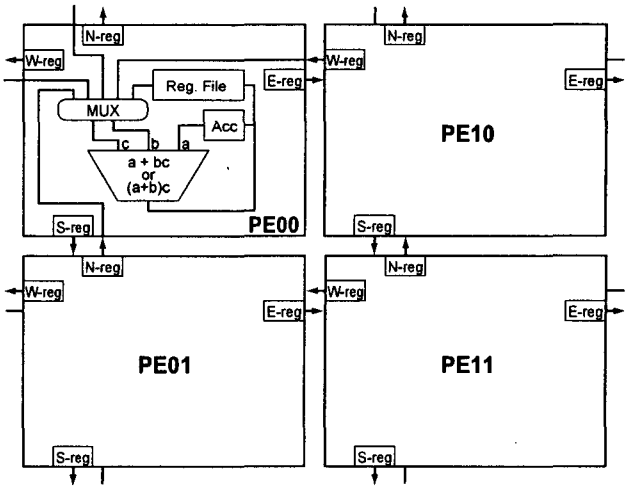

Fig. 2 Overview of the proposed array-based processor with 2x2 PEs.

$$\frac{\partial \boldsymbol{V}}{\partial t} + (\boldsymbol{V}\cdot\nabla)\boldsymbol{V} = -\nabla\varphi + \nu\nabla^2\boldsymbol{V} \tag{2}$$

where $\boldsymbol{V}$, φ, ν and t are a velocity vector($\equiv(u, v)$), a pressure($\equiv$P) divided by density, a kinematic viscosity and time, respectively. The fractional step method to solve these equations is composed of the following three steps.

Step1: Calculate a tentative velocity $\boldsymbol{V}^*$ with the equation of motion ignoring the pressure term.

$$\boldsymbol{V}^* = \boldsymbol{V}^n + \Delta t\left\{-(\boldsymbol{V}^n\cdot\nabla)\boldsymbol{V}^n + \nu\nabla^2\boldsymbol{V}^n\right\} \tag{3}$$

Step2: Calculate φ of the next time step with $\boldsymbol{V}^*$ by solving the following Poisson's equation.

$$\nabla^2\varphi^{n+1} = \frac{\nabla\boldsymbol{V}^*}{\Delta t} \tag{4}$$

Step3: Calculate the velocity of the next time step $\boldsymbol{V}^{n+1}$ with $\boldsymbol{V}^*$ and φ^{n+1}.

$$\boldsymbol{V}^{n+1} = \boldsymbol{V}^* - \Delta t\nabla\varphi^{n+1} \tag{5}$$

In the case of 2D flow, the difference scheme of the 2nd-order accuracy gives the following equations for the 2D staggered mesh shown in Fig. 1.

Step1:

$$u^*_{i,j} = au^n_{i,j} + b_u u^n_{i+1,j} + c_u u^n_{i-1,j} + d_u u^n_{i,j+1} + e_u u^n_{i,j-1} \tag{6}$$

$$v^*_{i,j} = av^n_{i,j} + b_v v^n_{i+1,j} + c_v v^n_{i-1,j} + d_v v^n_{i,j+1} + e_v v^n_{i,j-1} \tag{7}$$

where

$$a = 1 - 2\nu\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right),\quad b_u = \Delta t\left(\frac{\nu}{\Delta x^2} - \frac{u_{i,j}}{2\Delta x}\right),$$

$$c_u = \Delta t\left(\frac{\nu}{\Delta x^2} + \frac{u_{i,j}}{2\Delta x}\right),\quad d_u = \Delta t\left(\frac{\nu}{\Delta y^2} - \frac{v_{on\,u}}{2\Delta y}\right),\quad e_u = \Delta t\left(\frac{\nu}{\Delta y^2} + \frac{v_{on\,u}}{2\Delta y}\right)$$

$$\text{and}\quad v_{on\,u} = \frac{1}{4}\left(v_{i,j-1} + v_{i,j} + v_{i+1,j-1} + v_{i+1,j}\right).$$

b_v, c_v, d_v, e_v and $u_{on\,v}$ are calculated similarly.

Step2: (Gauss-Seidel method)

$$\varphi'_{i,j} = \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}\left(\frac{\varphi_{i+1,j} + \varphi'_{i-1,j}}{\Delta x^2} + \frac{\varphi_{i,j+1} + \varphi'_{i,j-1}}{\Delta y^2} - D_{i,j}\right) \tag{8}$$

where

$$D_{i,j} = \frac{1}{\Delta t}\left(\frac{u^*_{i,j} - u^*_{i-1,j}}{\Delta x} + \frac{v^*_{i,j} - v^*_{i,j-1}}{\Delta y}\right).$$

Step3:

$$u^{n+1}_{i,j} = u^*_{i,j} - \frac{\Delta t}{\Delta x}(\varphi'_{i+1,j} - \varphi'_{i,j}) \text{ and } v^{n+1}_{i,j} = v^*_{i,j} - \frac{\Delta t}{\Delta y}(\varphi'_{i,j+1} - \varphi'_{i,j}). \tag{9}$$

Once $u_{\text{on}\,v}$ and $v_{\text{on}\,u}$ are calculated at each grid point (i,j), Eqs.(6) and (7) of Step 1 can be written in the general form;

$$x^{\text{new}}_{i,j} = A + Bx_{i,j} + Cx_{i+1,j} + Dx_{i-1,j} + Ex_{i,j+1} + Fx_{i,j-1} \tag{10}$$

where $x_{i,j}$ is a certain value at grid (i, j), and A, B, C, D, E and F are values that can be calculated only with values at grid (i, j). Eqs.(8) and (9) of Steps 2 and 3 can also be formulated in Eq.(10). Even if we apply SOR method in Step 2, its iterative equation can be expressed in the same form.

Steps 1 to 3 written in the form of Eq.(10) are considered as a systolic algorithm due to their regularity and locality at each grid point. This means that the fractional step method can be implemented on an appropriate systolic array. Since calculation for grid (i, j) requires values of its adjacent grids, necessary communication is limited to the local area. Based on this key idea, we propose a dedicated processor relying upon a systolic array structure to accelerate the fractional step method.

Fig. 2 shows the basic architecture of the proposed processor for a 2D CFD solver. Note that we can easily expand an array into larger one of $n \times m$ PEs. The proposed processor has the systolic array structure with a 2D mesh network. Each PE has north(N-), south(S-), west(W-) and east(E-) registers connected to the four adjacent PEs, respectively. We call these registers *communication registers*. Once the PE writes an value onto these registers, the adjacent PEs can read and use them for computation at any time. The data-path of the PE shown in Fig. 2 is depicted simply for conceptual explanation. Each PE also has an arithmetic logic unit (ALU) with three inputs of a, b and c, which can compute $a + bc$ or $(a + b)c$ because such computations are frequently performed in the fractional step method. The register file of the PE stores necessary coefficients and variables. The accumulator, "Acc", is used to efficiently sum up the terms of Eq.(10). The ALU accepts the outputs of the register file or the values read from the adjacent PEs via the multiplexer, "MUX."

The register file is a local memory of the PE, and therefore the whole array can be considered as a computational memory composed of the distributed local memories. This is our systolic computational memory architecture where data stored in a memory are computed by the memory itself. This architecture inherently avoids the memory-processor bottleneck and has significant scalability; the larger array can scale up its computational performance with its overall memory bandwidth.

The calculation of Eq.(10) is simply performed on the systolic array. Firstly, the register files of PEs are initialised by using the communication registers as shift registers. Then, the terms of Eq.(10) are computed and summed up onto the accumulator one by one. This operation is synchronously performed by all the PEs while appropriate data are written to the communication registers at an appropriate time for communication. In the simple case that we can prepare the same number of PEs as grid points, we can easily map the calculation at a grid point to each PE. When we have less PEs than grid

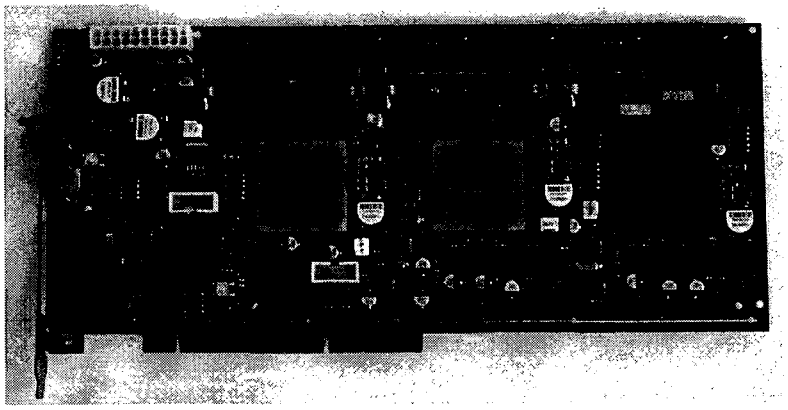

Fig. 3 PCI board with two Stratix II FPGAs (Courtesy of The DiNI group).

points, each PE takes charge of a grid block containing multiple grid points. For the former case, we have scheduled all computations and communications necessary for Steps 1 to 3 of the systolic algorithm.

We designed the PE for FPGA-based prototype implementation. Field-programmable gate array (FPGA) is a semiconductor device containing programmable logic components and programmable interconnects with memory elements and embedded arithmetic units. We made sure that our designed 32-bit single-precision floating-point adder of the PE can operate at more than 75MHz on ALTERA Stratix II FPGA (EP2S180-5). Since the hardware resource consumed for the adder is less than 0.5 % of the EP2S180 FPGA, we estimate that about 200 PEs on a prototype PCI board can be integrated with the two FPGA chips shown in Fig.3. The peak performance of the integrated 200 PEs, each of which has an adder and a multiplier operating at 75MHz, would reach 30Gflops of single-precision floating-point calculation. Although the Pentium 4 processor running at 3.8GHz has the peak performance of about 15Gflops for single-precision calculation, it's very difficult to maintain the peak performance for actual applications due to the limited memory-bandwidth. On the other hand, the distributed local memories of our systolic-based processor allow the ALUs to fully be utilized. If we could implement our proposed processor by using the same technology and hardware resources as such a high-end microprocessor, much higher frequency and integration of PEs would lead to tremendous performance of difference computation with a single chip.

REFERENCES

[1] J. Kim and P. Moin, "Application of a fractional-step method to incompressible Navier-Stokes," Journal of Computational Physics, Vol.59, pp. 308–323, 1985.

[2] T. Hauser, "A Flow Solver for a Reconfigurable FPGA-Based Hypercomputer," 43rd AIAA Aerospace Sciences Meeting and Exhibit, AIAA–2005–1382, 2005.

[3] H.T. Kung, "Why Systolic Architecture?," IEEE Computer, Vol.15, No.1, pp.37 –46, 1982

[4] K. Sano, C. Takagi, R. Egawa, K. Suzuki and T. Nakamura, "A Systolic Memory Architecture for Fast Codebook Design based on MMPDCL Algorithm," Proceedings of the International Conference on Information Technology Coding and Computing (ITCC2004), pp.572–578, 2004.

[5] K. Sano, C. Takagi and T. Nakamura, "Systolic Computational Memory Approach to High-Speed Codebook Design," Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT2005), pp.334–339, 2005.