Optimization of FEM solver for heterogeneous multicore processor Cell

Noriyuki Kushida[1]

[1] Center for Computational Science and e-system
Japan Atomic Energy Research Agency
6-9-3 Higashi-Ueno, Taito-ku, Tokyo 110-0015, JAPAN
Kushida.noriyuki@jaea.go.jp

In this study, I introduce a novel implementation method of finite element method in order to reduce the memory access the Cell which is a kind of heterogeneous multicore processor. Thanks to the invention of multicore processor, the computational performance per one silicon chip is rapidly enlarging, therefore, the relative memory bandwidth, which is growing slower than computational performance and usually limits the performance of scientific computing code, is decreasing. The Cell, which is developed by Sony, Toshiba, and IBM, is the one of multicore processor, and has relatively high computational performance than the other commodity processors. In order to hide the poorness of memory bandwidth, small but high speed memory, which is called Local store(LS), is set just under the synergetic processing unit(SPU) in the Cell architecture. Since higher computational performance of the Cell is obtained when the less access to the main memory is achieved, we introduced the method which decreased the access ratio to the main memory at the expense of computational effort. As a result, we observed better performance by comparing with ordinal finite element Poisson solver, which run on PowerPC processing unit(PPU) of the Cell.

Optimization of FEM solver for heterogeneous multicore processor Cell

Noriyuki Kushida (JAEA)

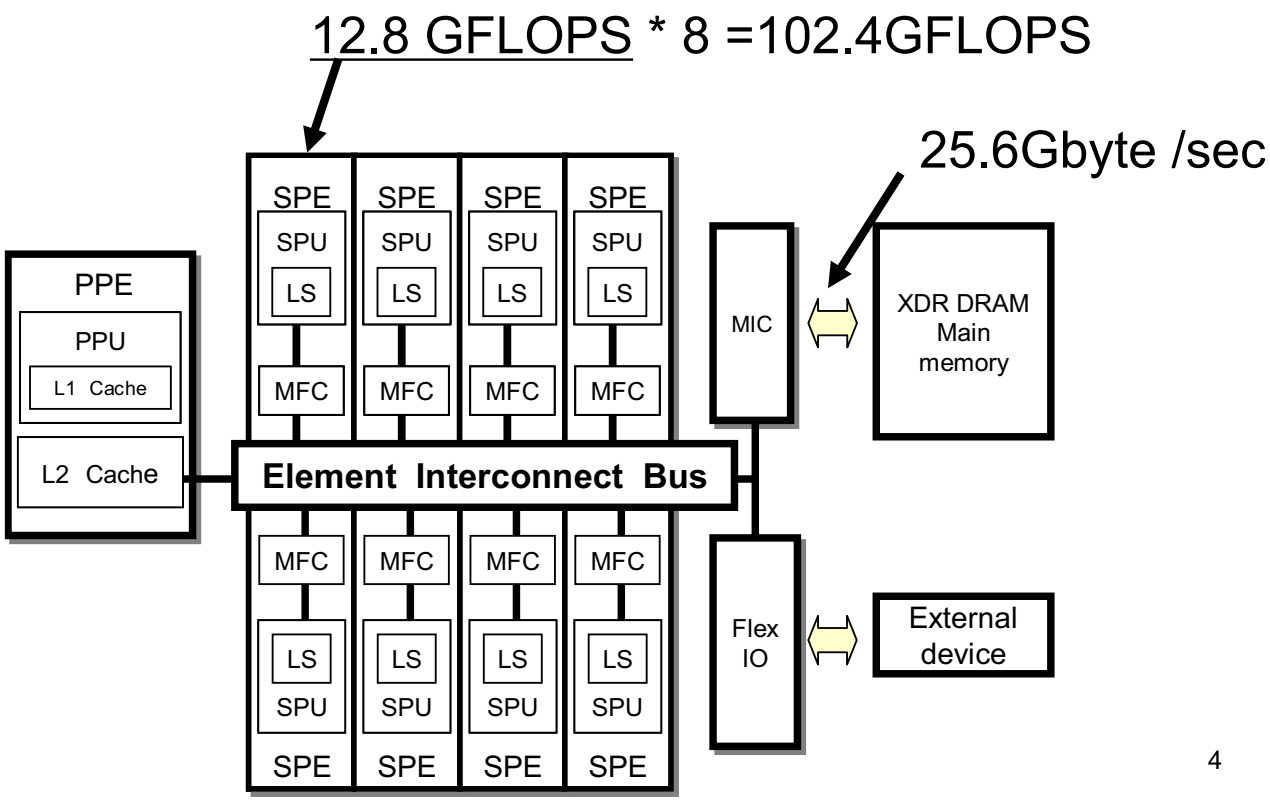
Background

- Processing capacity of computer chip is still growing by the Moor's law
  - Especially, invention of multicore processor helps growth
- But, growth of memory bandwidth to one chip is lesser than that of processing capacity.
- Cache memory or similar kind of feature is set in order to increase effective memory bandwidth.
  - Not effective for most scientific computing,,,

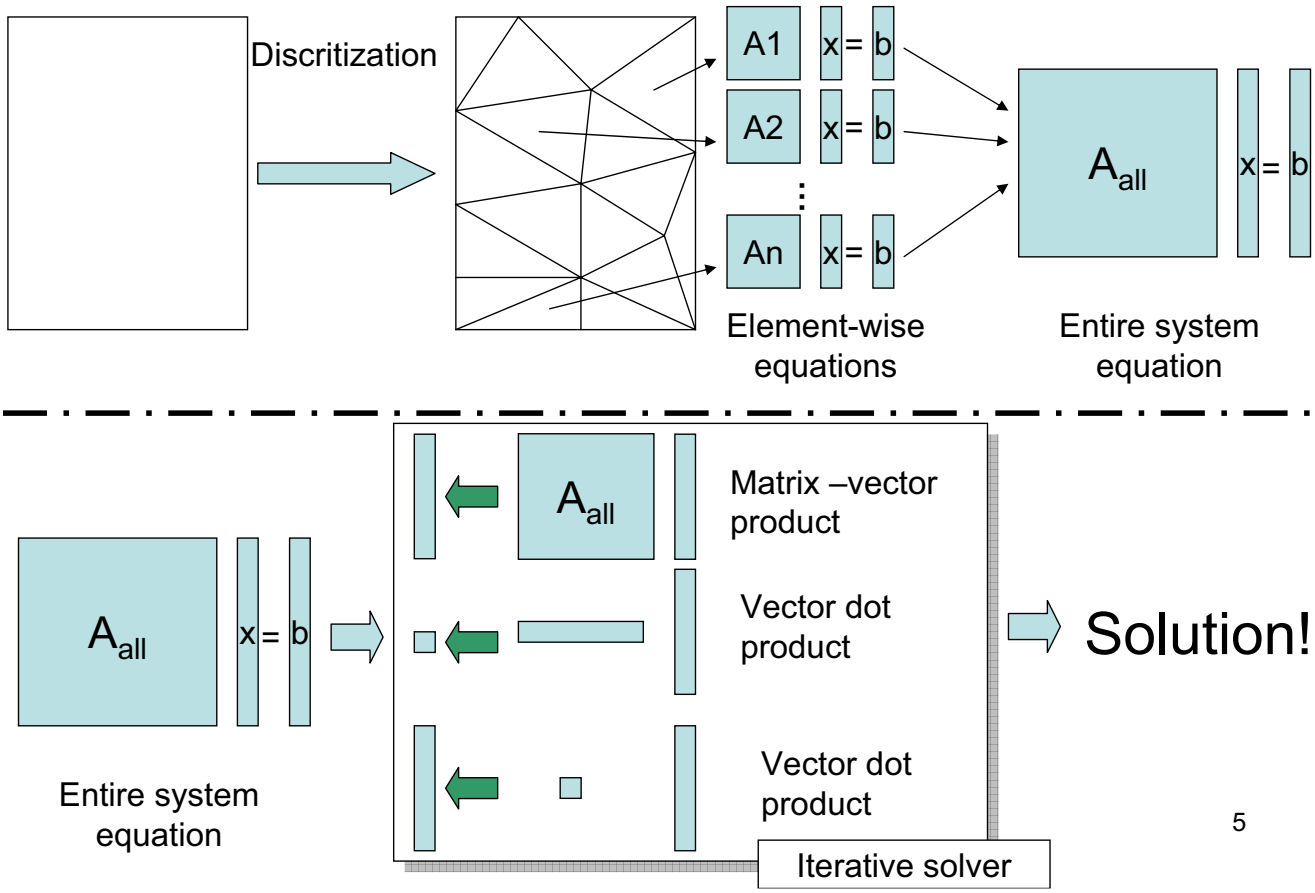
Objectives

- New methodology reducing the memory access, since the relative memory bandwidth will tend to be narrow for near future.
- Sony, Toshiba, and IBM provided novel processor chip named Cell
- Cell is
  - A kind of heterogeneous multicore processor
  - Has a cache like memory whose behavior can be strictly controlled by the programmer
- In this study, I introduced memory access saving implementation of finite element method, and the effectiveness was investigated by comparing the performance of usual FEM implementation.

The cell processor (PowerXCell 8i)



Naive FEM solver



Conjugate gradient

1. Let $\mathbf{x}^{(0)}$ be the initial approximation
2. $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. for $i = 0, 1, 2, 3, \ldots,$ until convergence do
4. $\quad \rho^{(i)} = (\mathbf{r}^{(i)}, \mathbf{r}^{(i)})$
5. $\quad$ if $i = 0$ then
6. $\quad\quad \mathbf{p}^{(i+1)} = \mathbf{r}^{(i)}$
7. $\quad$ else
8. $\quad\quad \beta^{(i)} = \rho^{(i)} / \rho^{(i-1)}$
9. $\quad\quad \mathbf{p}^{(i+1)} = \mathbf{p}^{(i)} + \beta^{(i)} \mathbf{r}^{(i)}$
10. $\quad$ end if
11. $\quad \mathbf{q}^{(i+1)} = \mathbf{A}\mathbf{p}^{(i+1)}$
12. $\quad \alpha^{(i)} = \rho^{(i)} / \left(\mathbf{p}^{(i+1)}, \mathbf{q}^{(i+1)}\right)$
13. $\quad \mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)} \mathbf{p}^{(i+1)}$
14. $\quad \mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} - \alpha^{(i)} \mathbf{q}^{(i+1)}$
15. end for

Matrix-vector multiplication

The coefficient matrix is only required at these two lines, and only with matrix – vector multiplication form.

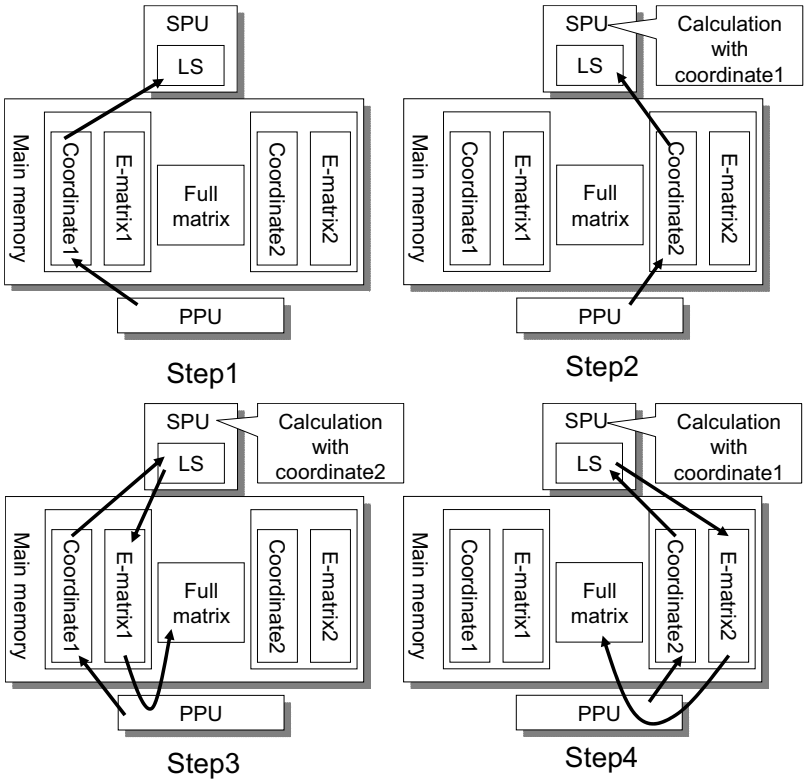
Memory access

- Following two parts are the heaviest memory access part in the FEM
- Matrix construction
  - Search location of matrix component
  - Load value from entire matrix
  - Add element wise matrix
  - Store to global matrix
- Matrix – vector product
  - Load a row component
  - Search location and load vector component

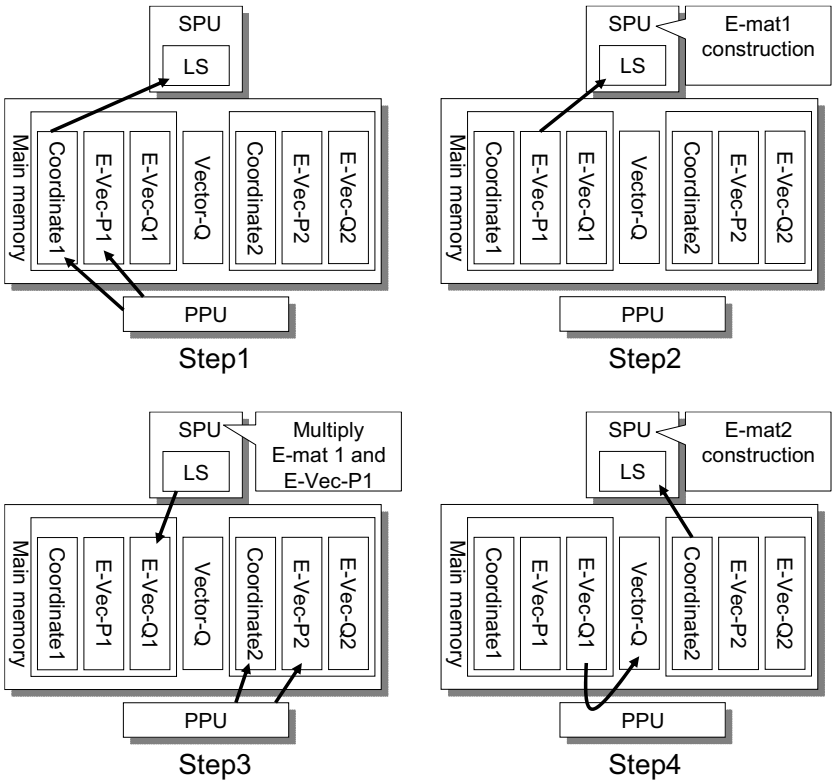
Strategy to access save

- Reduce the amount of memory access
- Hide memory access time behind the calculation
  - Double buffering
- If the matrix-vector multiplication can be done by only with element-wise procedure, we avoid the global memory access.
  - We only need to access quite localized memory.

Implementation on the Cell – Matrix construction -

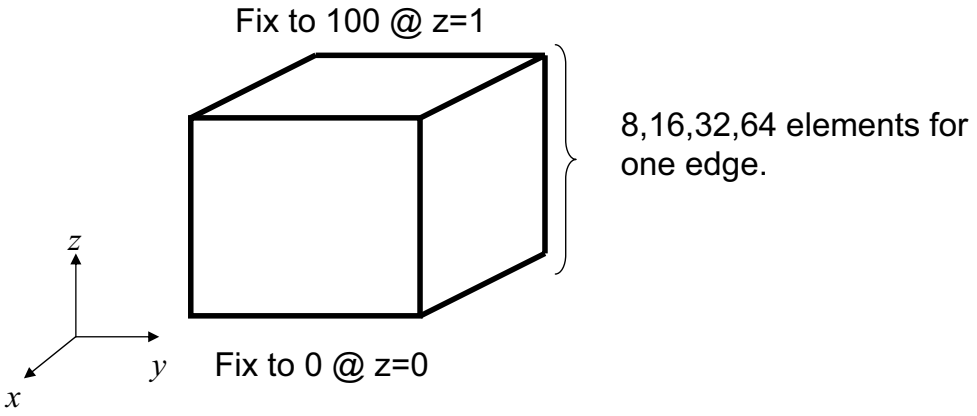


Implementation on the Cell
- Element wise matrix vector product -



Speed up measurement

- 3-dimensional cubic analysis domain
- Descritized with 8^3, 16^3, 32^3, 64^3 elements



Measurement conditions

- Following four implementations are performed on the PowerXCell 8i processor
  - Normal FEM only on PPU
  - Normal FEM using SPU's
  - Access saving FEM only on PPU
  - Access saving FEM using SPU's

Results

Implementation	Calculation time(sec)			
	8^3	16^3	32^3	64^3
PPU-normal	1.78E-02	3.85E-01	9.30E+00	2.62E+02
PPU-access saving	8.09E-02	1.28E+00	2.06E+01	3.01E+02
SPU-normal	9.87E-02	1.67E+00	2.99E+01	5.63E+02
SPU-access saving	6.60E-03	9.17E-02	1.97E+00	2.90E+01

- Access saving FEM using SPU is the best for every problem size.
- Normal FEM only on PPU is the second best.
- Normal FEM using SPU is the worst for every problem size.

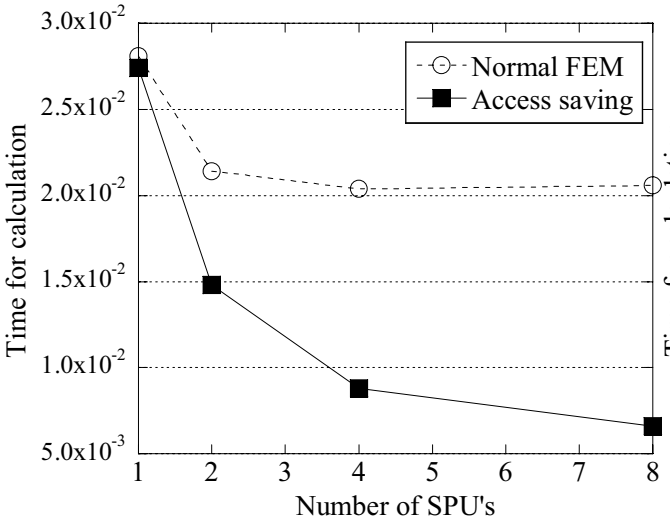
- Access saving FEM does not reduce the amount of calculation (FLOP)
- When naïve implementation is applied, using SPU's slow down the performance

Parallel efficiency measurement

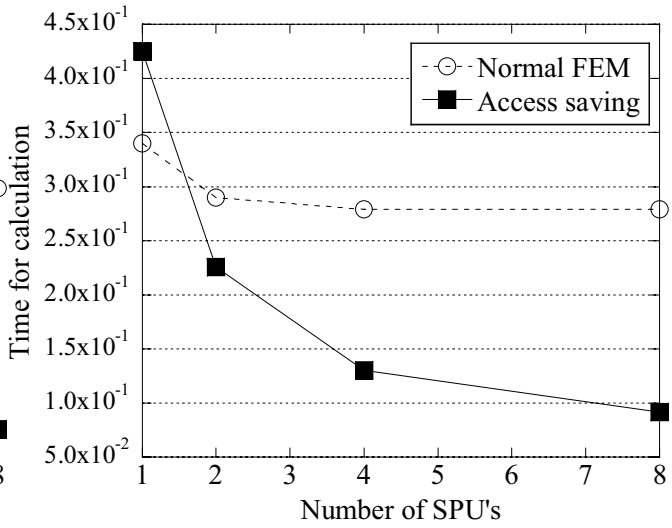
- When SPU's are used, we can observe the performance behavior by the change of FLOP / bandwidth ratio

- Following two implementations are used for measurement
  - Normal FEM using SPU's
  - Access saving FEM using SPU's

Parallel efficiency 1/2

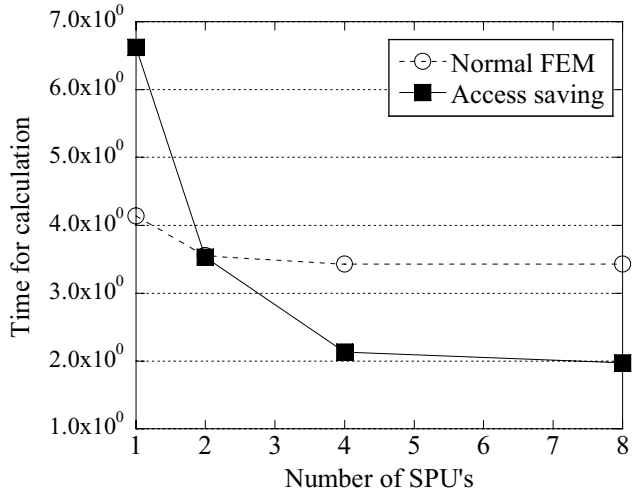


8^3 elements

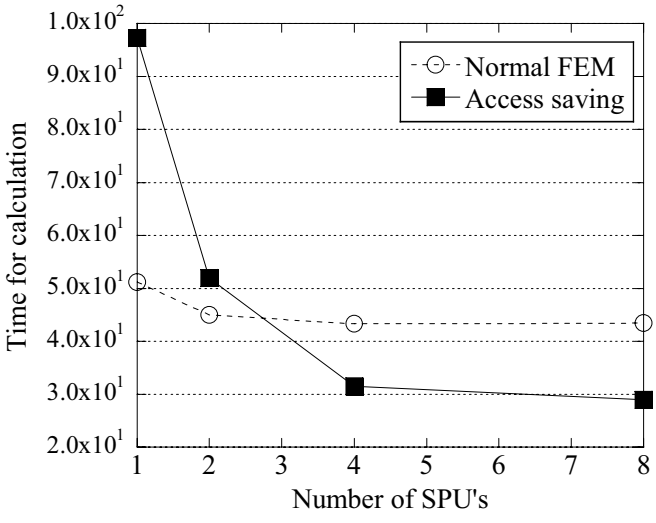


16^3 elements

Parallel efficiency 2/2



32^3 elements



64^3 elements

Conclusions

- I introduced memory access saving FEM implementation, and it was implemented on the Cell processor
- By comparing the performance of normal FEM implementation and memory access saving FEM, following conclusion can be obtained
  - Although memory access saving FEM requires much floating point operations than normal FEM, memory access saving FEM was the fastest implementation in this study.
  - Thanks to the less memory access, parallel efficiency of memory access saving FEM was better than normal FEM.

Fin