Parallel ADI Method for Parabolic Problems on GP-GPU

Hyoungseok Chu

Together with Hyoseop Lee and Dongwoo Sheen

Interdisciplinary Program
in Computational Science and Technology
Seoul National University
http://cst.snu.ac.kr

October 17, 2009

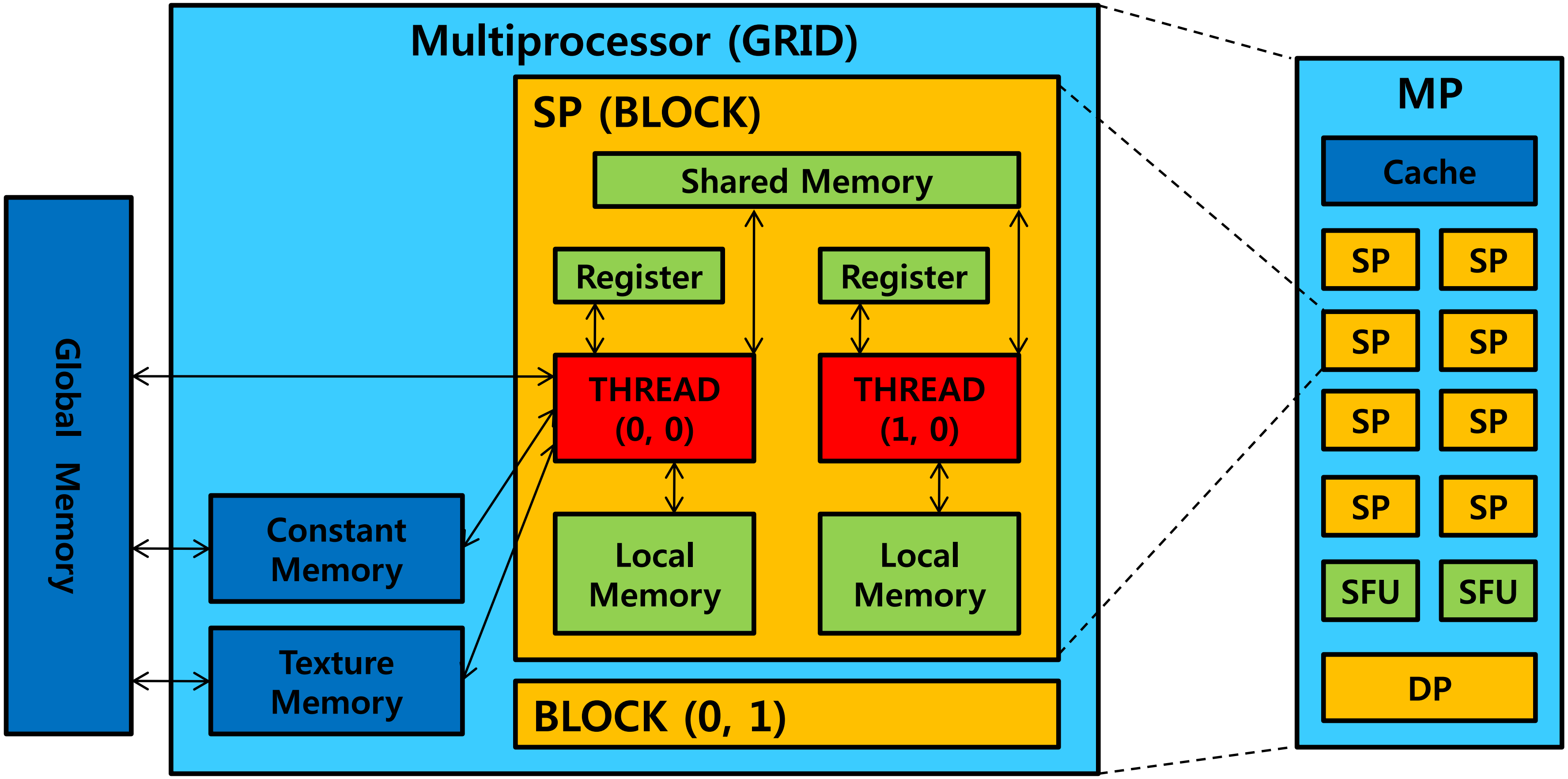
Outline

1. **Introduction**
   - GP-GPU
   - Target Problem
2. **Implementation**
   - Parallelization
3. **Numerical Result**
   - Examples
4. **Conclusion**

Outline

1. **Introduction**
   - GP-GPU
   - Target Problem
2. **Implementation**
   - Parallelization
3. **Numerical Result**
   - Examples
4. **Conclusion**

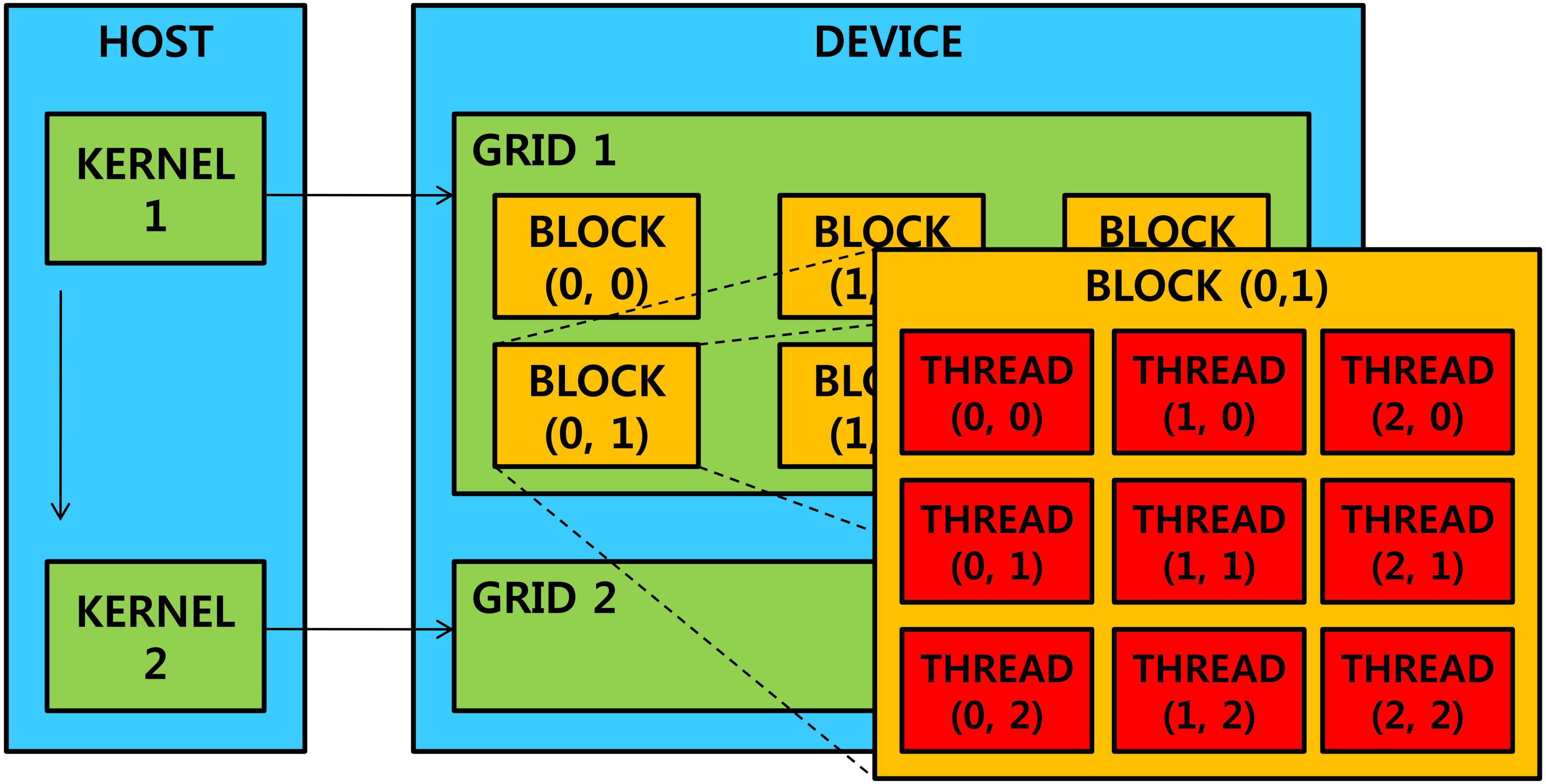
GPU-Architecture (1/2)



Source : CUDA Programming Guide (revised)

GP-GPU

GPU-Architecture (2/2)



Source : CUDA Programming Guide (revised)

GPU Capability (1/2)

Advantages

- Powerful single precision operation
  - perform almost **1 Tera FLOPS** at saxpy operation
  - cheaper price than CPU system
- **Light weight threading, Shared memory**
- Dealing with **Graphical process**
  - support DirectX, OpenGL
- Programmable in C, Fortran, etc : **CUDA**

GPU Capability (2/2)

Disadvantages

- Bottleneck on **memory bandwidth** (PCI-E x16 : 4GB/s)
  - communication bandwidth between host and device
  - example : using large scale memory
- Block communication
  - **can not share** information between blocks
  - example : dot product
- Branch diverges
  - command will be **delayed, pending** others
  - example : case statement

3-D Parabolic Equation

Three dimensional parabolic equation

$$u_t(x,t) - \sum_{i,j=1}^{3} \nu_{ij} u_{x_i x_j}(x,t) + \sum_{i=1}^{3} c_i u + \gamma u(x,t) = 0$$

where $(x,t) \in \Omega \times (0,T]$ and

$$au(x,t) + b\frac{\partial u}{\partial n}(x,t) = f(x,t) \qquad (x,t) \in \partial\Omega \times (0,T]$$

$$u(x,t) = u_0(x) \qquad (x,t) \in \Omega \times \{0\}$$

Alternating Direction Implicit Method

Craig-Sneyd ADI method (with mixed derivative)

SET NT to be the number of time step

DO $n=0$ to NT-1

$$(1-\theta\nu_{11}r\delta_{x_1}^2)u_{(1)}^{n+1(1)} = Au^n$$

$$(1-\theta\nu_{22}r\delta_{x_2}^2)u_{(1)}^{n+1(2)} = u^{n+1(1)} - \theta\nu_{22}r\delta_{x_2}^2 u^n$$

$$(1-\theta\nu_{33}r\delta_{x_3}^2)u_{(1)}^{n+1} = u^{n+1(2)} - \theta\nu_{33}r\delta_{x_3}^2 u^n$$

$$(1-\theta\nu_{11}r\delta_{x_1}^2)u^{n+1(1)} = Au^n + \lambda B(u_{(1)}^{n+1} - u^n)$$

$$(1-\theta\nu_{22}r\delta_{x_2}^2)u^{n+1(2)} = u^{n+1(1)} - \theta\nu_{22}r\delta_{x_2}^2 u^n$$

$$(1-\theta\nu_{33}r\delta_{x_3}^2)u^{n+1} = u^{n+1(2)} - \theta\nu_{33}r\delta_{x_3}^2 u^n$$

END DO

Parameters

Algorithm Flow

```
DO t = initial time, final time * 2, time step

  IF λ == 0 AND iteration == even continue
  set up the tridiagonal matrix system
  solve the tridiagonal matrix system with respect to x axis
  get the first step vector u^{n+1(1)} and change axis

  set up the tridiagonal matrix system
  solve the tridiagonal matrix system with respect to y axis
  get the second step vector u^{n+1(2)} and change axis

  set up the tridiagonal matrix system
  solve the tridiagonal matrix system with respect to z axis
  get the final step vector u^{n+1} and change axis
  IF λ ≠ 0 AND iteration == odd THEN
     assign u^{n+1} to u_{(1)}^{n+1}

END DO
```

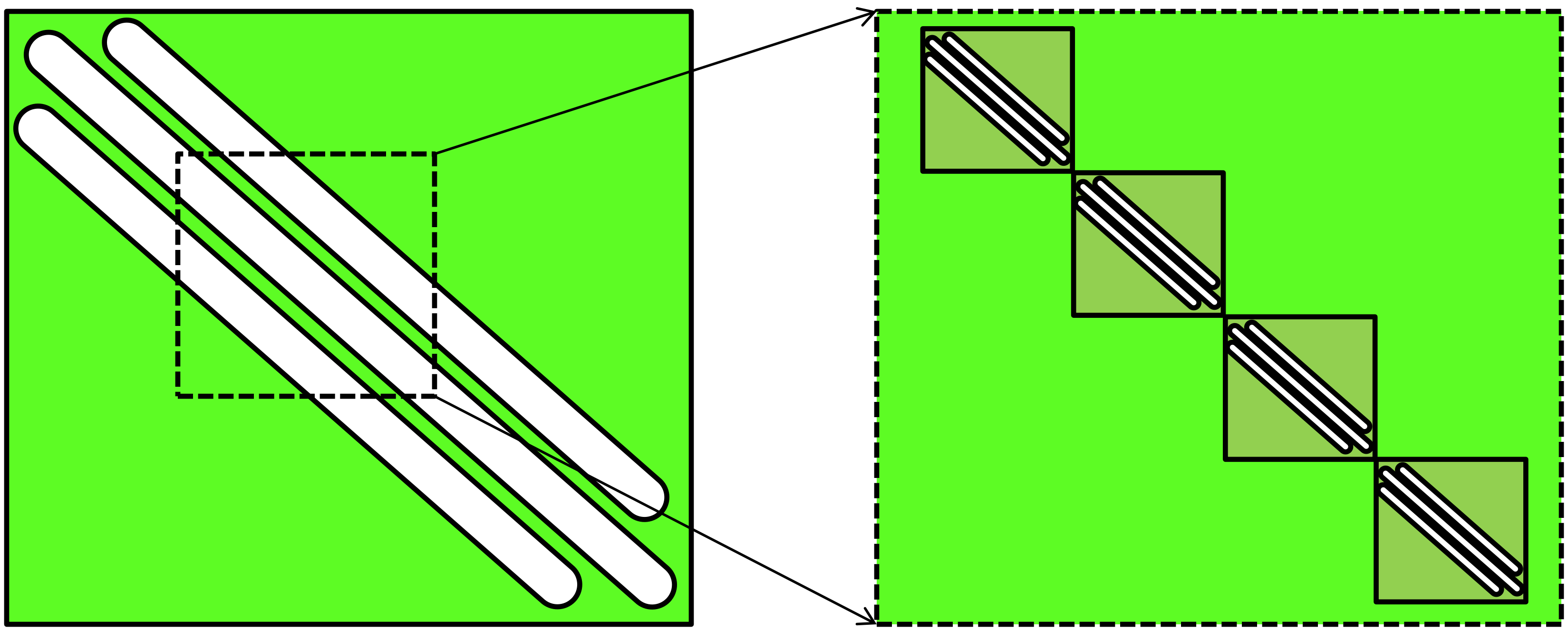
Outline

1. Introduction
   - GP-GPU
   - Target Problem
2. **Implementation**
   - **Parallelization**
3. Numerical Result
   - Examples
4. Conclusion

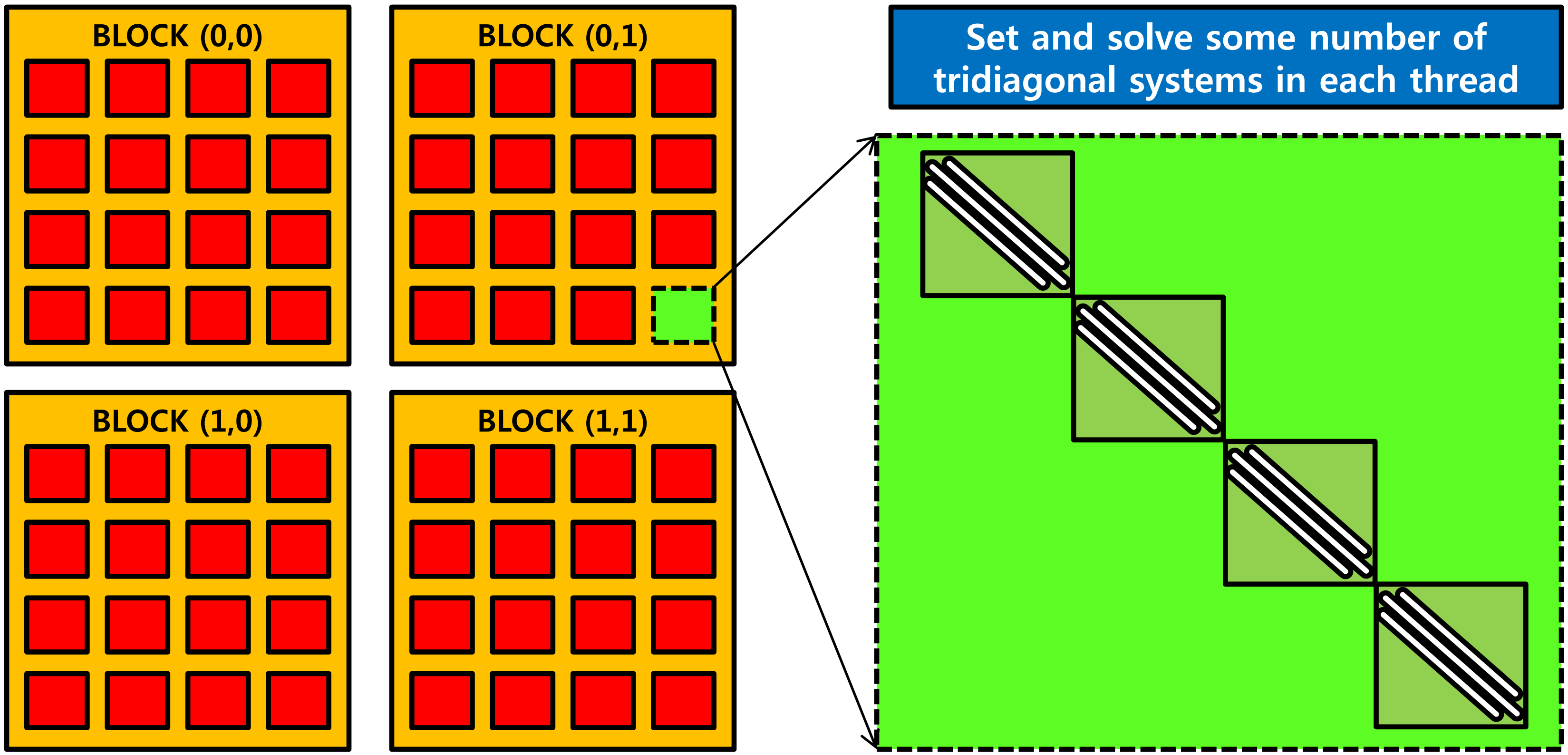
The composition of Matrix (from ADI method)

N^3 X N^3 Tridiagonal matrix

N^2 number of independent tridiagonal matrix



Tridiagonal systems assigned to each threads



Job Distribution

Assign tridiagonal systems to thread

- available thread number of GPU device : T
- the block size : $m \times n$
- the total number of threads : $T_{total} = T \times m \times n$
- determine the number of tridiagonal system per thread
  - the basic number : N^2 / T_{total}
  - modulus $M = N^2 \bmod T_{total}$ will be added M number of threads

Outline

1. **Introduction**
   - GP-GPU
   - Target Problem
2. **Implementation**
   - Parallelization
3. **Numerical Result**
   - Examples
4. **Conclusion**

Problems

3D parabolic equation

Coefficients

$$\gamma = 0 \qquad c_1 = -0.4 \qquad c_2 = -0.7 \qquad c_3 = -0.5$$

$$(\nu_{ij}) = \begin{bmatrix} 0.5 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \qquad domain = (0,2)^3 \times (0,1]$$

$$u(x_1, x_2, x_3, t) = \frac{\exp\left(-\frac{(x_1 - c_1 - 0.5)^2}{\nu_{11}(1+4t)} - \frac{(x_1 - c_2 - 0.5)^2}{\nu_{22}(1+4t)} - \frac{(x_1 - c_3 - 0.5)^2}{\nu_{33}(1+4t)}\right)}{(1+4t)^{3/2}}$$

Comparison

Comparison

- GPU : NVIDIA GTX285 **Spec**
- CPU : Intel Dual Core E2150 1.6GHz
- Reduction rate

$$\log_2 \frac{\| u_{\Delta x, \Delta t} - u_{exact} \|_{L^2(\Omega)}}{\| u_{\frac{\Delta x}{2}, \frac{\Delta t}{2}} - u_{exact} \|_{L^2(\Omega)}}$$

- Speed up

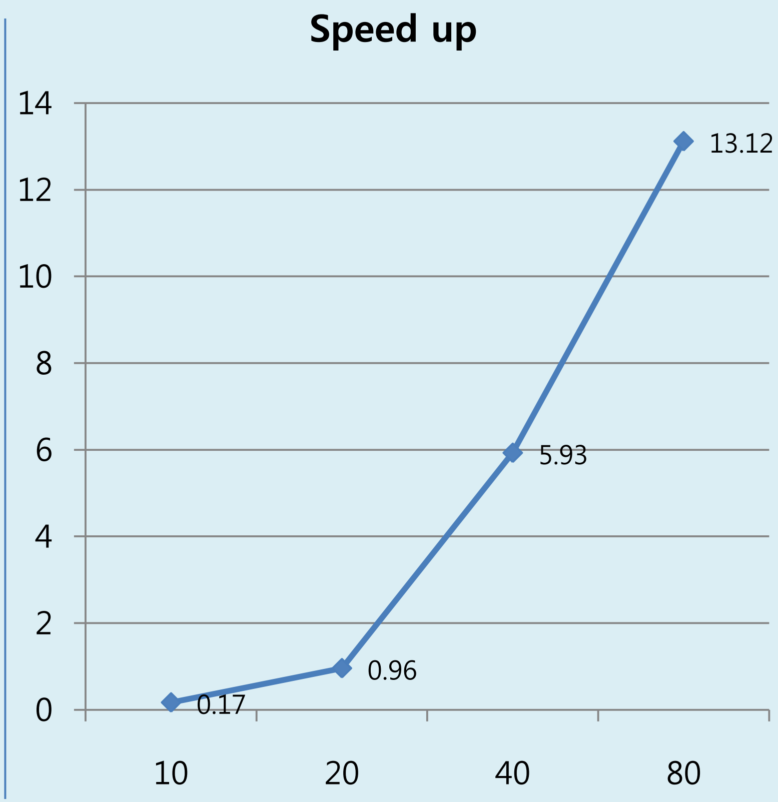
$$\frac{\text{Time consumption on CPU}}{\text{Time consumption on GPU}}$$

Example

Results (1/3)

Result chart (single, θ=0.5, λ=0.0)

nt,nx	Device	Time (sec)	L2-Error	Reduction	Speedup
20,10	CPU	0.03	2.81E-03		
	GPU	0.18	2.81E-03		0.17x
40,20	CPU	0.24	1.17E-03	1.26	
	GPU	0.25	1.17E-03	1.26	0.96x
80,40	CPU	3.32	5.73E-04	1.03	
	GPU	0.56	5.74E-04	1.03	5.93x
160,80	CPU	49.99	2.88E-04	0.99	
	GPU	3.81	2.91E-04	0.98	**13.12x**

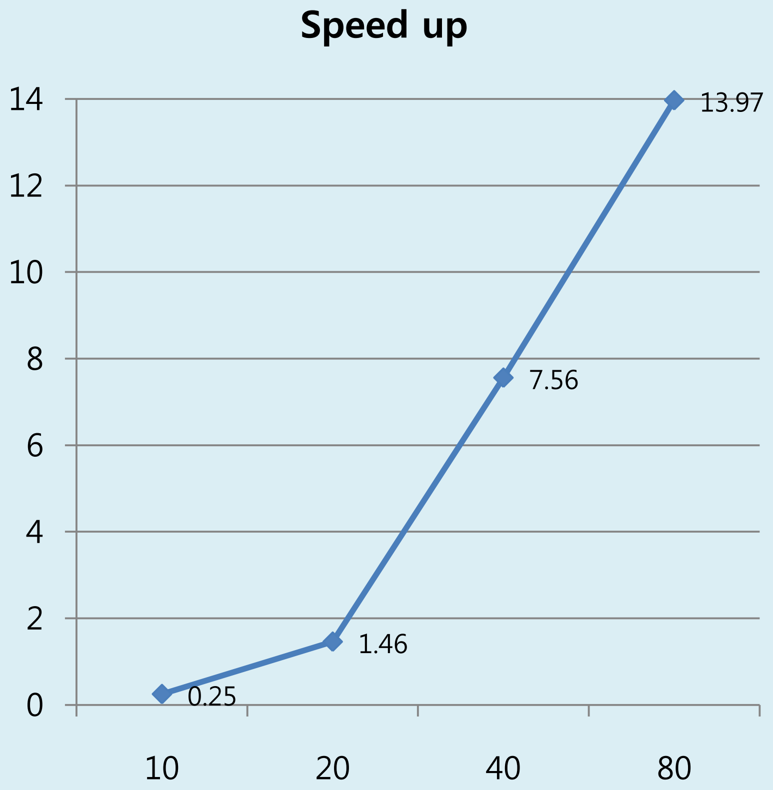


Example

Results (2/3)

Result chart (single, θ=0.5, λ=0.5)

nt,nx	Device	Time (sec)	L2-Error	Reduction	Speedup
20,10	CPU	0.05	2.03E-03		
	GPU	0.20	2.03E-03		0.25x
40,20	CPU	0.51	5.02E-04	2.02	
	GPU	0.35	5.02E-04	2.02	1.46x
80,40	CPU	7.18	1.25E-04	2.01	
	GPU	0.95	1.25E-04	2.00	7.56x
160,80	CPU	118.24	3.10E-05	2.01	
	GPU	8.49	3.20E-05	1.97	**13.97x**

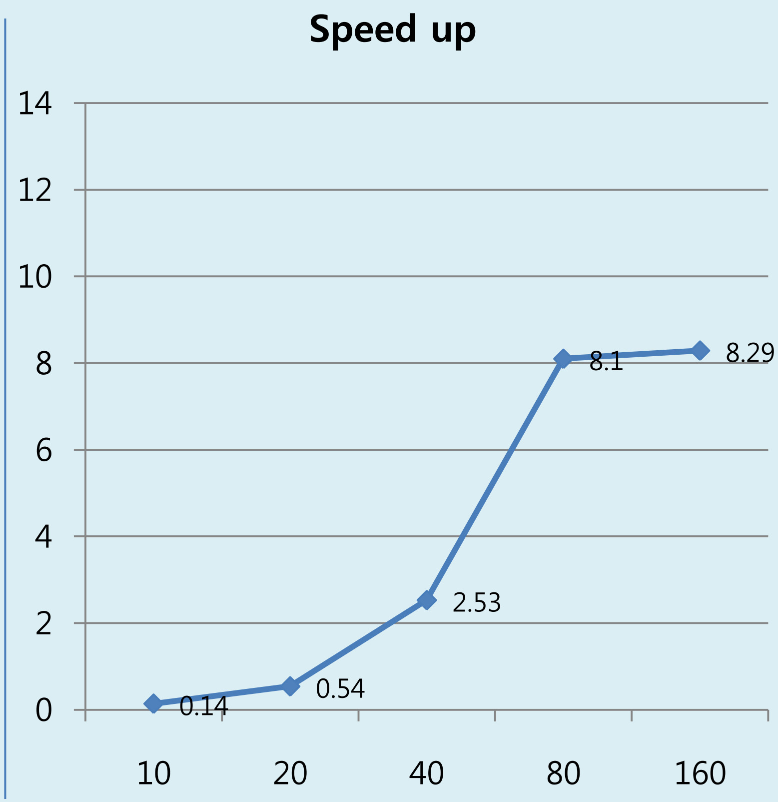


Example

Results (3/3)

Result chart (double, θ=0.5, λ=0.5)

nt,nx	Device	Time (sec)	L2-Error	Reduction	Speedup
20,10	CPU	0.06	2.03E-03		
	GPU	0.42	2.03E-03		0.14x
40,20	CPU	0.52	5.02E-04	2.02	
	GPU	0.96	5.02E-04	2.02	0.54x
80,40	CPU	7.77	1.25E-04	2.01	
	GPU	3.07	1.25E-04	2.01	2.53x
160,80	CPU	126.49	3.12E-05	2.00	
	GPU	15.61	3.12E-05	2.00	**8.10x**
320,160	CPU	2141.38	7.78E-06	2.00	
	GPU	258.19	7.78E-06	2.00	**8.29x**



Outline

1. **Introduction**
   - GP-GPU
   - Target Problem
2. **Implementation**
   - Parallelization
3. **Numerical Result**
   - Examples
4. **Conclusion**

Conclusion

- **Speed up**
  - 13 times in single, 8 times in double
- **Applicable problems**
  - Black-Scholes Equation
- **Hard to optimize**
  - memory issue, threads per register
- **Fermi Architecture**
  - more powerful in double precision

Thank you for listening

Any Questions?

Contact : hschu@snu.ac.kr

Alternating Direction Implicit Method

Craig-Sneyd ADI method (parameters)

$$A = 1 + r(1-\theta)v_{11}\delta_{x_1}^2 + r\sum_{i=2}^{3} v_{ii}\delta_{x_i}^2 + \frac{1}{2}\sum_{i=2}^{3}\sum_{j=1}^{i-1} v_{ij}\delta_{x_i x_j} + \frac{1}{2}r\Delta x\sum_{i=3}^{3} c_i\delta_{x_i} + \gamma\Delta t$$

$$B = \frac{1}{2}r\sum_{i=2}^{3}\sum_{j=1}^{i-1} v_{ij}\delta_{x_i x_j} + \frac{1}{2}r\Delta x\sum_{i=1}^{3} c_i\delta_{x_i} + \gamma\Delta t$$

$$\delta_{x_1}u_{i,j,k} = u_{i+1,j,k} - u_{i-1,j,k}$$

$$\delta_{x_1}^2 u_{i,j,k} = u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}$$

$$\delta_{x_1 x_2}u_{i,j,k} = u_{i+1,j,k} - u_{i+1,j-1,k} - u_{i-1,j+1,k} + u_{i-1,j-1,k}$$

Back

GPU Specification (1/2)

GTX 285

- 30 multiprocessors
  - 8 SP / multiprocessor
  - 1 DP / multiprocessor
- 1.29 GHz / SP or DP
- Single precision FLOPS
  - 933 GFLOPS (at peaks)
- Double precision FLOPS
  - 78 GFLOPS (at peaks)

Source : NVIDIA website

GPU Specification (2/2)

GTX 285 Computing Capability

- the number of threads / block : 512
- the number of threads / warp : 32
- the number of registers / multiprocessor : 16384
- shared memory / multiprocessor : 16Kbyte
- active threads / multiprocessor : 1024
- active blocks / multiprocessor : 8
- active warps / multiprocessor : 32

Back

Source : CUDA Programming Guide