Tsunami simulation on FPGA/GPU and its analysis based on Statistical Model Checking

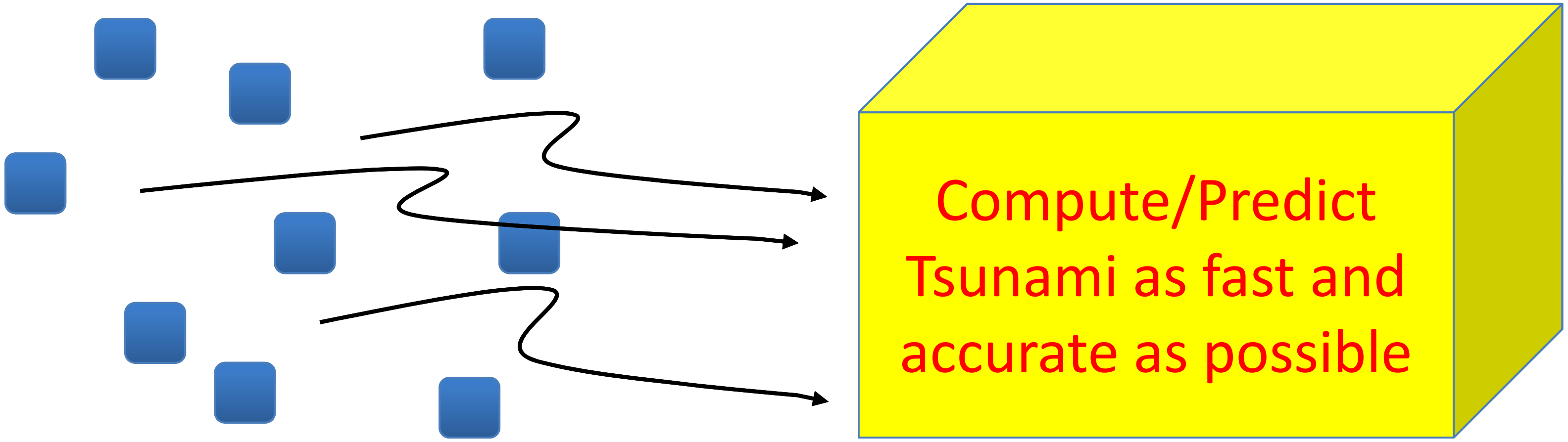
Masahiro Fujita
VLSI Design and Education Center (VDEC)
University of Tokyo

Outline

- What Tsunami simulation means in this talk
- Acceleration with FPGA/GPU
  - Based on stream processing (pipelining) with loop unrolling
  - Based on parallel processing for decomposed regions
- (Formal verification of those implementation)
  - (Equivalence checking between FPGA/GPU implementation and the original program in C/Fortran)
  - Just show our strategy
- Statistical model checking
  - On software in Fortran
  - Acceleration with FPGA/GPU

Motivation

- Based on the values of many earthquake sensors (wired/wireless), compute how Tsunami wave will propagate
- Goal: Realize supercomputer level performance in Tsunami simulation with FPGA/GPU



Compute/Predict Tsunami as fast and accurate as possible

Earthquake sensors geographically distributed

Generate initial wave from sensor data

Propagate wave by numerically solving partial differential equations

Tsunami simulation

- Tsunami simulation algorithm: Find solutions of fluid dynamics equations
  - Law of Conservation of Mass
  - Law of Conservation of Momentum with and without bottom friction
- Solved with known boundary conditions and bathymetric input of the region
- Here the above is processed by numerically solving sets of partial differential equations with finite difference methods

Partial differential equations to be solved

ὴ = vertical displacement of water above still water
D= Total water depth = h+ὴ
g = Acceleration due to gravity
A = horizontal eddy viscosity current
τ = friction along x or y direction
M = water flux discharge along X direction
N = water flux discharge along Y direction

$$\frac{\partial \eta}{\partial t} + \frac{\partial M}{\partial x} + \frac{\partial N}{\partial y} = 0$$

Mass conservation

$$\frac{\partial M}{\partial t} + \frac{\partial}{\partial x}\left(\frac{M^2}{D}\right) + \frac{\partial}{\partial y}\left(\frac{MN}{D}\right) + gD\frac{\partial \eta}{\partial x} + \frac{\tau_x}{\rho} = A\left(\frac{\partial^2 M}{\partial x^2} + \frac{\partial^2 M}{\partial y^2}\right)$$

$$\frac{\partial N}{\partial t} + \frac{\partial}{\partial x}\left(\frac{MN}{D}\right) + \frac{\partial}{\partial y}\left(\frac{N^2}{D}\right) + gD\frac{\partial \eta}{\partial y} + \frac{\tau_y}{\rho} = A\left(\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 N}{\partial y^2}\right)$$

Momentum equations along X-axis and Y-axis respectively without bottom friction

Reference: Tsunami Modeling Manual by Prof Nobuo Shuto

Here we use a simplified model: Linear one (valid if sea depth is large enough)

- **Shallow Water Theory** **(Long Wave Theory)**

$$\frac{\partial M}{\partial t}+\frac{\partial M}{\partial x}+\frac{\partial N}{\partial y}=0$$

(Mass Conservation)

$$\frac{\partial M}{\partial t}+\frac{\partial}{\partial x}\left(\frac{M^2}{D}\right)+\frac{\partial}{\partial y}\left(\frac{MN}{D}\right)+gD\frac{\partial \eta}{\partial x}+\frac{gn^2M}{\partial D^{\frac{7}{3}}}\sqrt{M^2+N^2}=0$$

$$\frac{\partial M}{\partial t}+\frac{\partial}{\partial x}\left(\frac{MN}{D}\right)+\frac{\partial}{\partial y}\left(\frac{N^2}{D}\right)+gD\frac{\partial \eta}{\partial y}+\frac{gn^2M}{\partial D^{\frac{7}{3}}}\sqrt{M^2+N^2}=0$$

(Momentum Conservation)

$\eta : waveheight$ $D : depth$ $g : gravity$ $n : Manning$ $M, N : flaxofx, y$

- **Linear Long Wave Theory**

$$\frac{\partial \eta}{\partial t}+\frac{\partial M}{\partial x}+\frac{\partial N}{\partial y}=0$$

(Mass Conservation)

$$\frac{\partial M}{\partial t}+gh\frac{\partial \eta}{\partial x}=0, \qquad \frac{\partial N}{\partial t}+gh\frac{\partial \eta}{\partial y}=0$$

(Momentum Conservation)

Finite difference methods

- Solution of mass conservation equation based on finite difference method
  - Z(i,j,t+1) = Z(I,j,t) – (dt/dx)*(M(I,j,t)-M(i-1,j,t)+N(i,j,t)-N(i,j-1,t))

$$\frac{\partial M}{\partial t}+\frac{\partial M}{\partial x}+\frac{\partial N}{\partial y}=0$$

Where

i, j = x, y coordinate

Z(i,j,t) = Water Surface level at time t

H(i,j,t) = Still water depth at time t

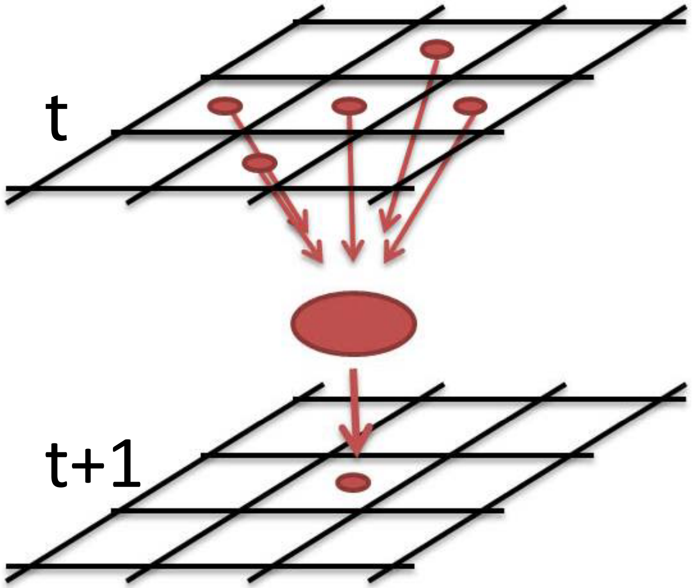
dt = temporal step

dx = spatial step

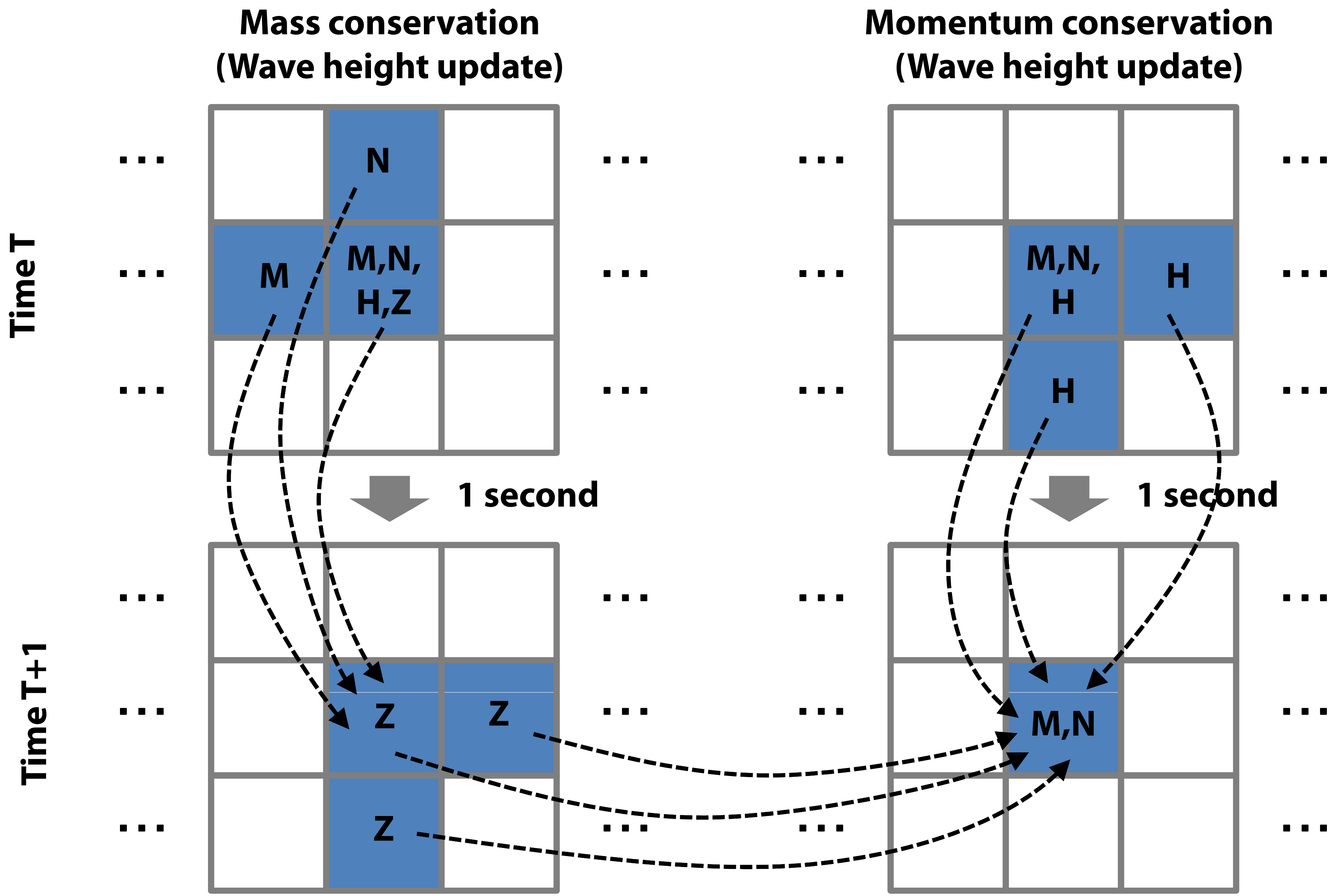
M(i,j,1) = water flux discharge along x-axis at time t

N(i,j,1) = water flux discharge along y-axis at time t

Z(i,j,2) = water surface level at time t+dt

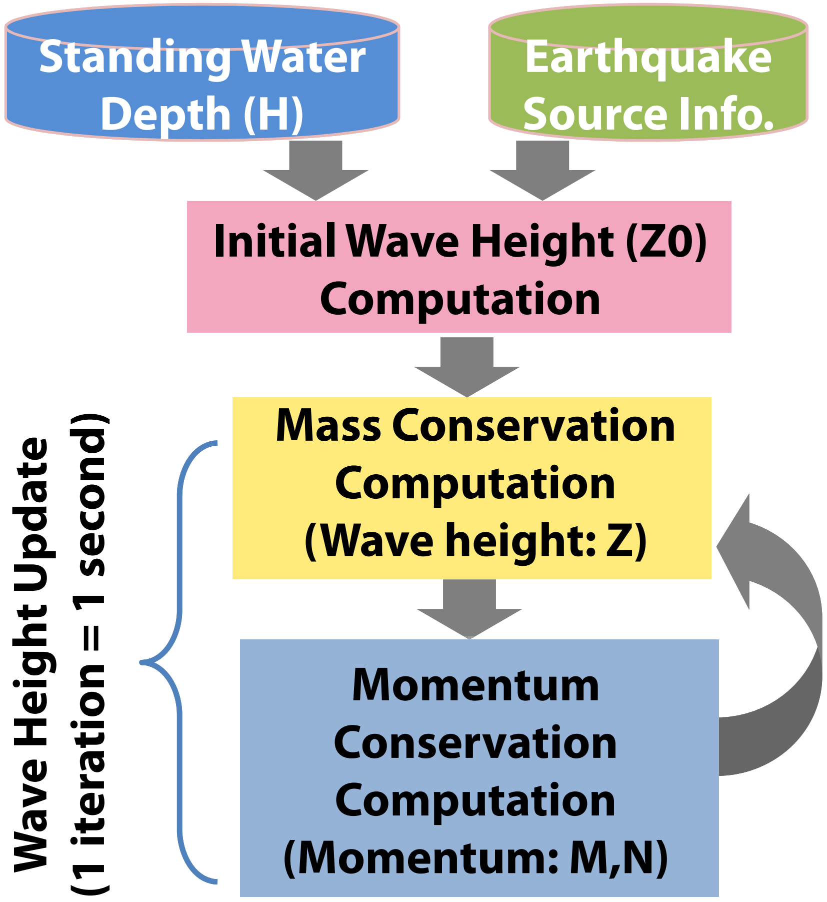


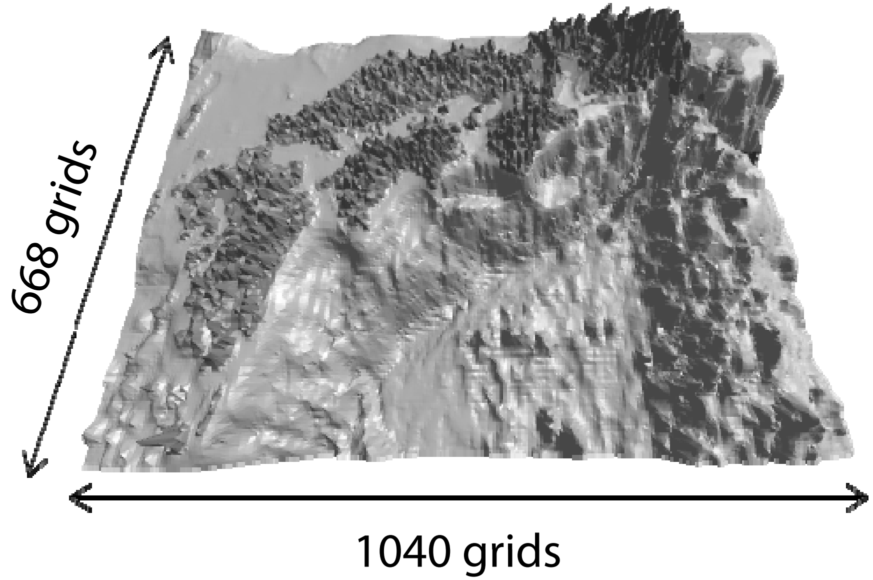
Wave Height Computation



Target Tsunami Simulator

- "TUNAMI N1" program in FORTRAN
  - Developed by Tohoku University

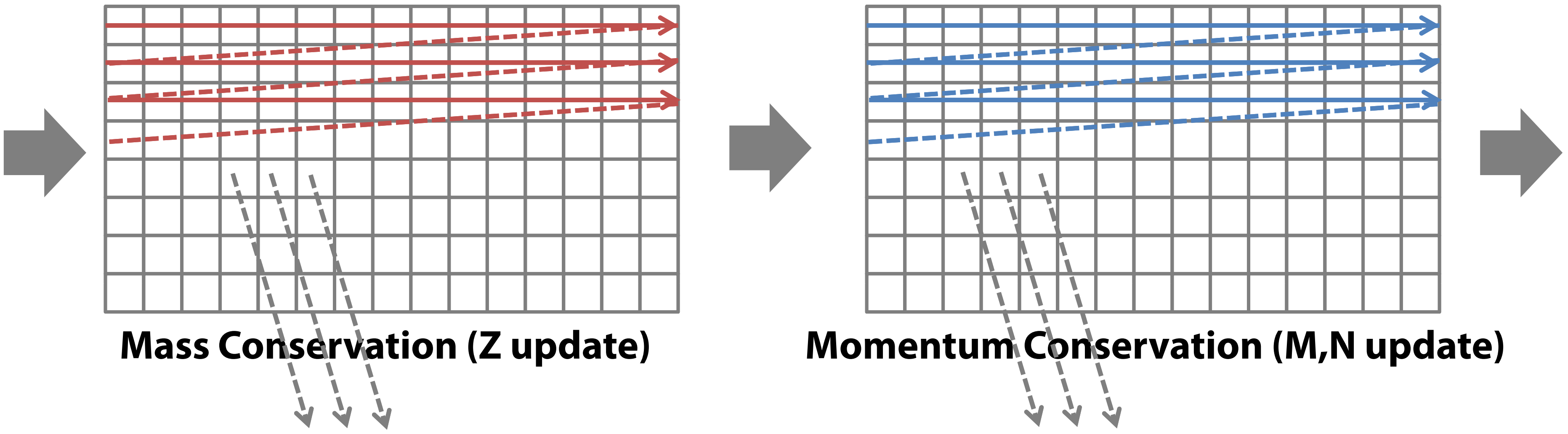




Standing water depth H
(1 grid: 1km x 1km)

C Implementation (base program)

- Mass and Momentum functions are computed alternatively
  - Each function raster-scans the grids
- Since there is no data dependency between the computations at grids, they can be parallelized



Mass Conservation (Z update)

Momentum Conservation (M,N update)

No dependency (Parallelizable)

No dependency (Parallelizable)

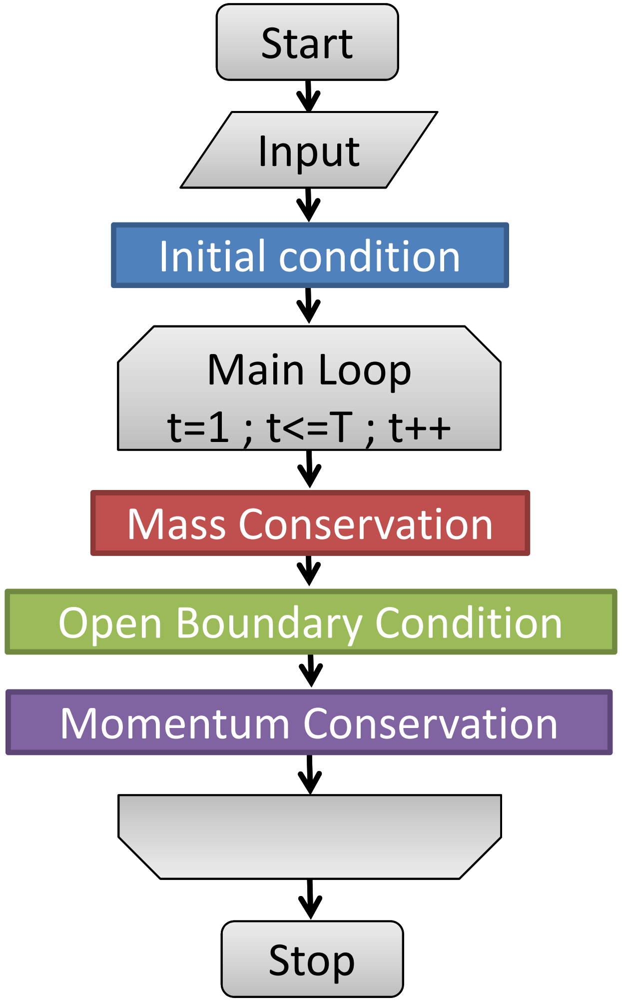
Speed of N1 simulation program

- Original Fortran program has been manually converted into C
  - C is 4 times faster than Fortran in our environment
  - C version is the base simulator
- Size of simulation area
  - Grid width: 1[km]
  - Numbers of grids: 1040*668
- Simulated time
  - 1 time step = 1 sec
  - 7,200 steps computed (2 hours)

- Tsunami simulation time on Intel microprocessor (i7@2.93GHz, single core)

  → 78.7 sec

Profiling of Computation time of the software



Start

Input

Initial condition

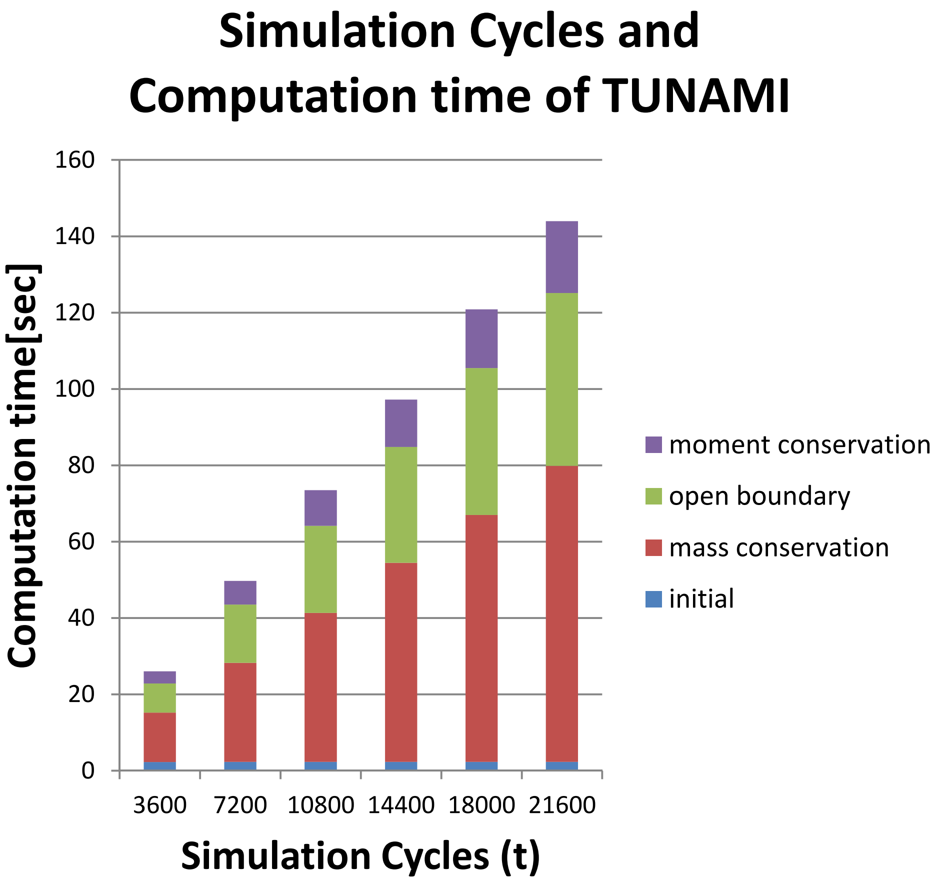
Main Loop
t=1 ; t<=T ; t++

Mass Conservation

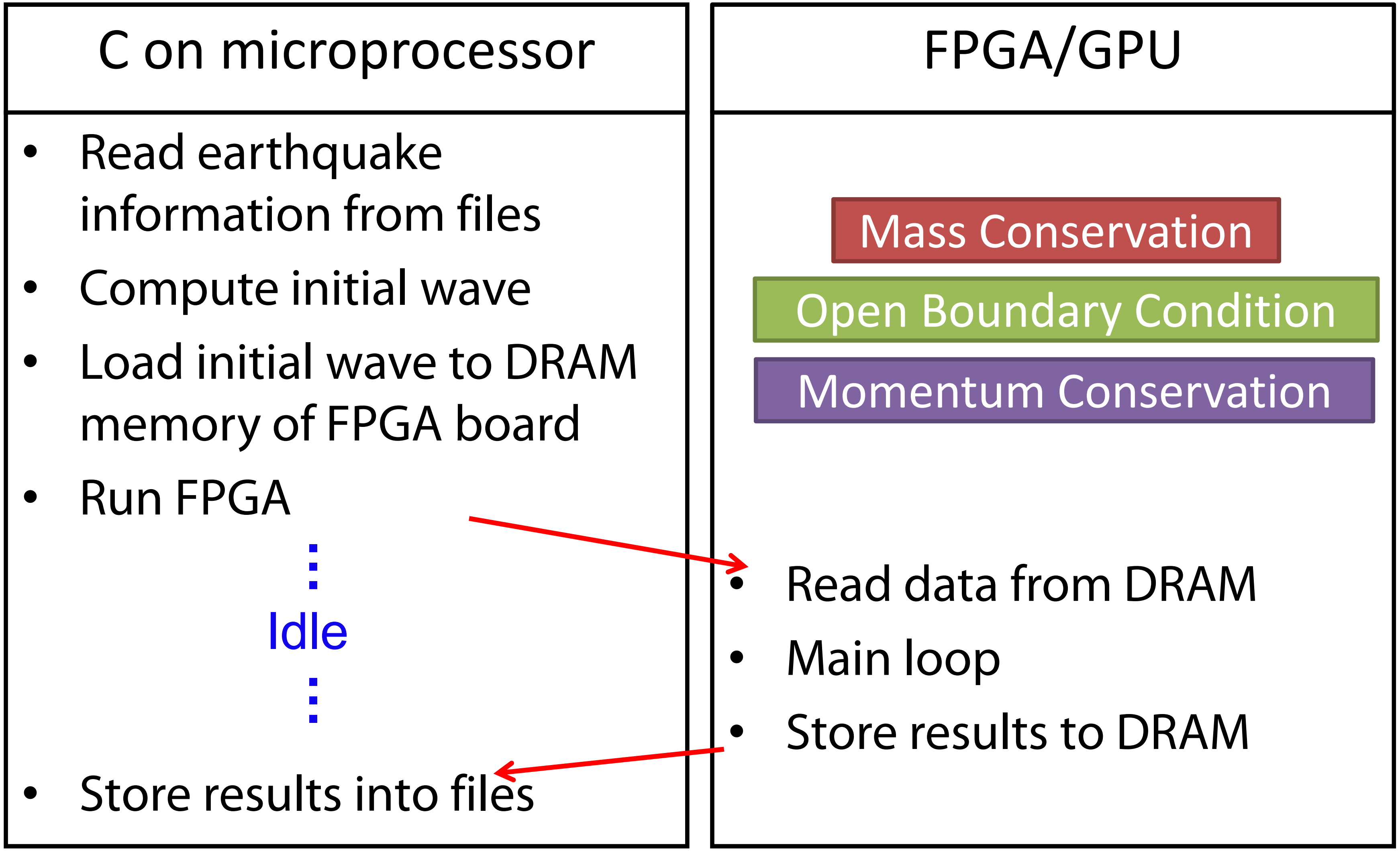
Open Boundary Condition

Momentum Conservation

Stop



Co-execution



C on microprocessor

- Read earthquake information from files
- Compute initial wave
- Load initial wave to DRAM memory of FPGA board
- Run FPGA

⋮

Idle

⋮

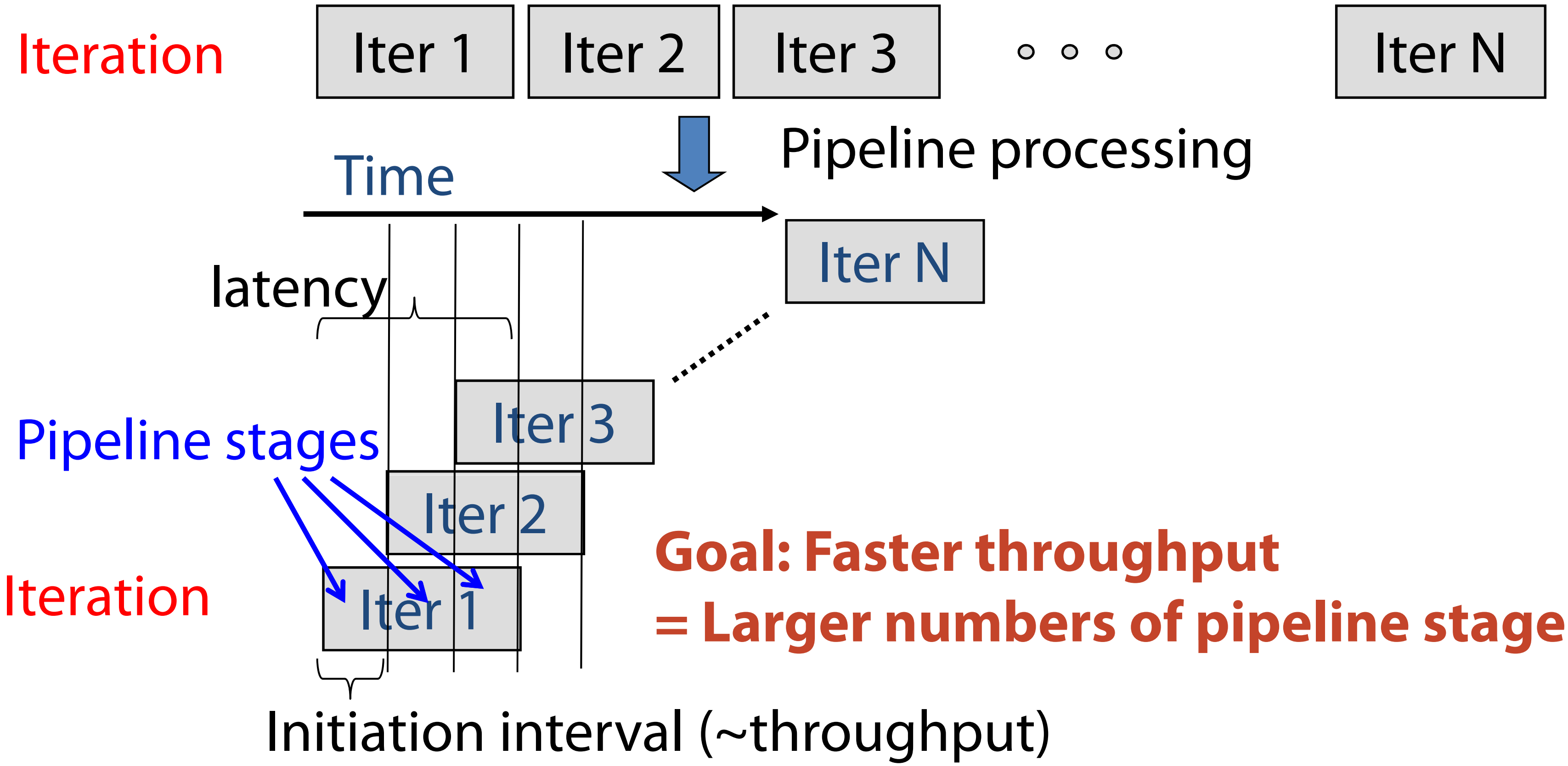
- Store results into files

FPGA/GPU

- Read data from DRAM
- Main loop
- Store results to DRAM

Pipeline processing for higher throughput

- Latency
  - After receiving input, how many cycles are required to generate its output
- Throughput
  - How frequently input data can be processed



Iteration

Iter 1 | Iter 2 | Iter 3 | ∘ ∘ ∘ | Iter N

Pipeline processing

Time

latency

Iter N

Iter 3

Pipeline stages

Iter 2

Iteration

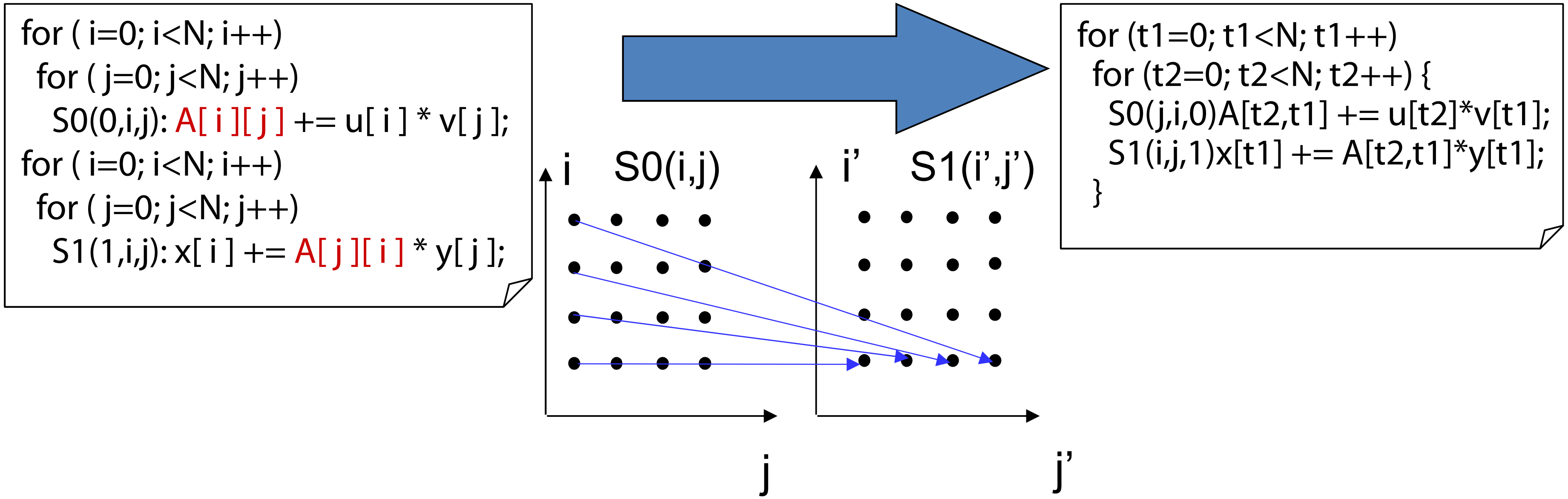
Iter 1

Initiation interval (~throughput)

Goal: Faster throughput
= Larger numbers of pipeline stages

Typical way for larger pipelines

- Usually each loop becomes one pipeline
  - Multiple loops should be merged as much as possible
- Number of pipeline stages depends on the length of each iteration
  - Better to have larger loops
- Various loop optimizations have been proposed
  - Formal analysis becomes possible with such transformations



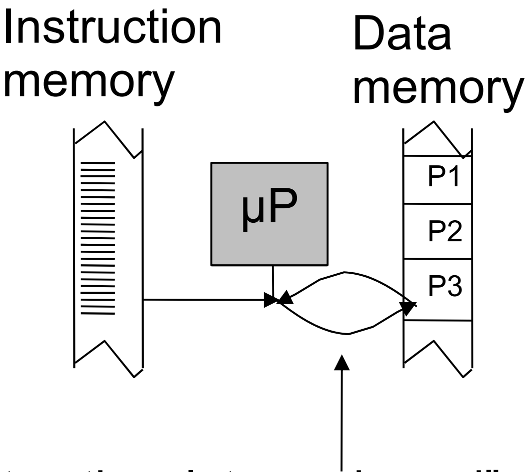
```
for ( i=0; i<N; i++)
 for ( j=0; j<N; j++)
  S0(0,i,j): A[ i ][ j ] += u[ i ] * v[ j ];
for ( i=0; i<N; i++)
 for ( j=0; j<N; j++)
  S1(1,i,j): x[ i ] += A[ j ][ i ] * y[ j ];
```

```
for (t1=0; t1<N; t1++)
 for (t2=0; t2<N; t2++) {
  S0(j,i,0)A[t2,t1] += u[t2]*v[t1];
  S1(i,j,1)x[t1] += A[t2,t1]*y[t1];
 }
```

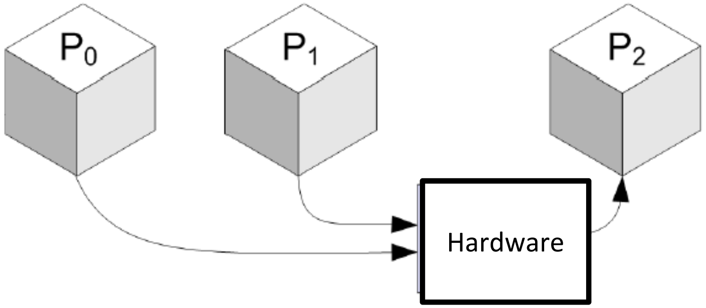
[Pluto 08] *U. Bondhugula, et al. “"A Practical and Automatic Polyhedral Program Optimization System,” in ACM PLDI'08, 2008*

Transforming latency-based to throughput-based computation

- Stream based programming
  - Communication/buffering becomes explicit
  - Easier for formal analysis as well
- Works for both FPGA and GPU
  - And also for many-cores



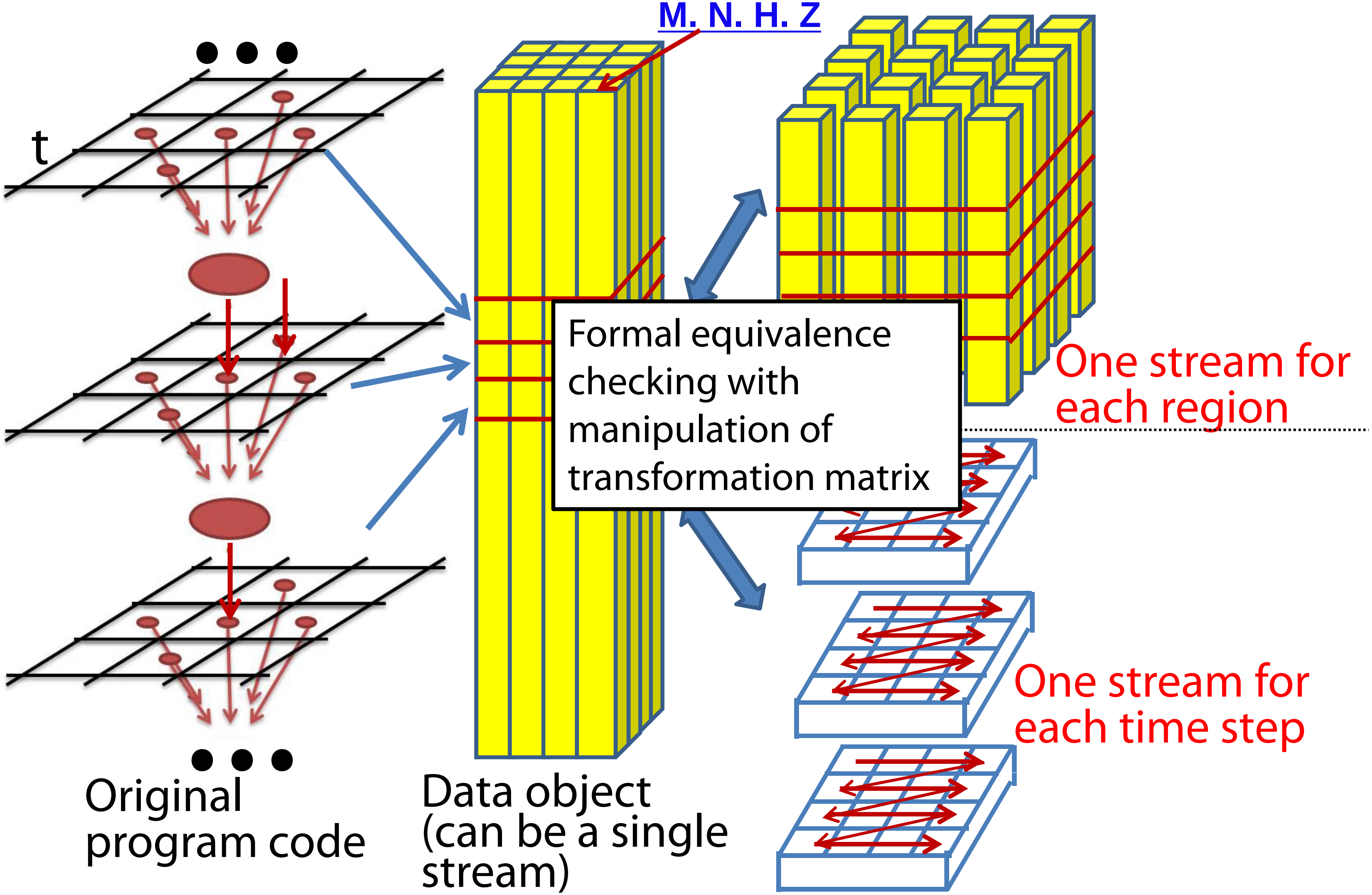
“instruction, latency-based”



“data, throughput-based”

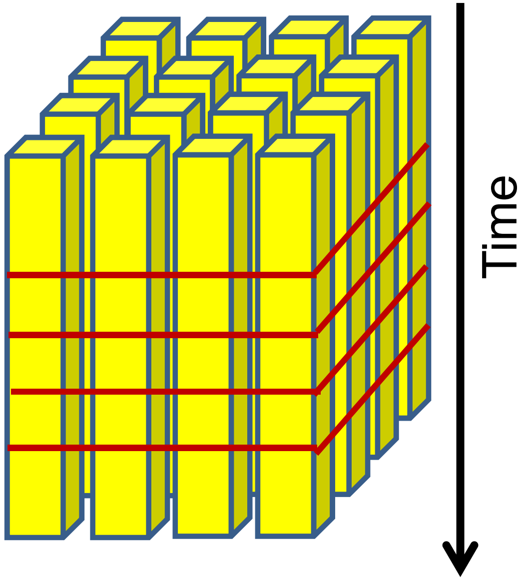
Introducing streams

- Steam = Sequence of data, functions, or combined



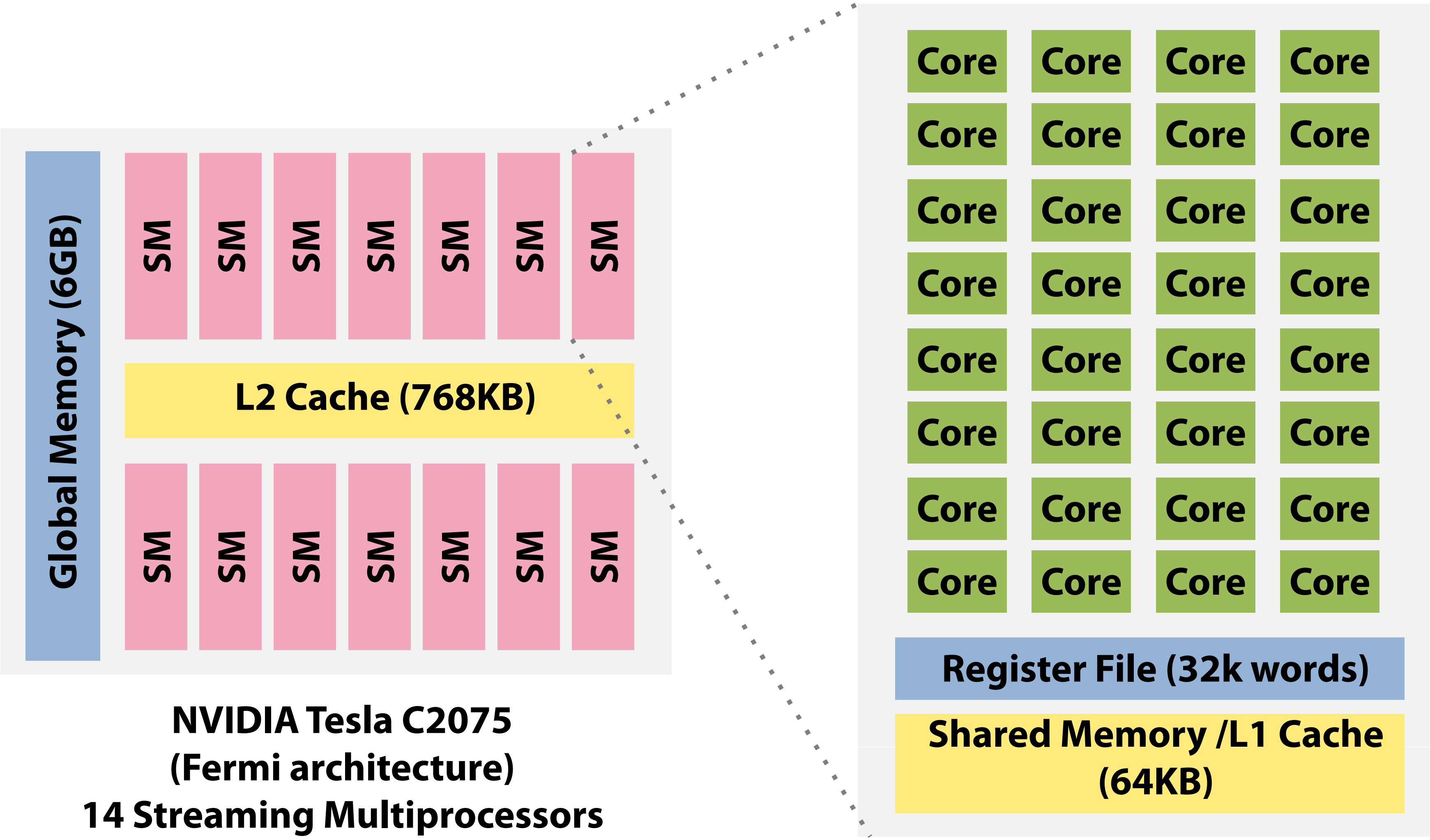
Strategy for GPU implementation

- As shown earlier, stream is based on each region
  - Easier and more efficient for GPU
  - But depend on memory access architecture of the target GPU systems



- Essentially area where Tsunami should be simulated is decomposed into a set of small regions
  - Each core of GPU is in charge of one region
  - Straight forward parallelism
  - Pipelined computation inside each core

Target GPGPU Architecture

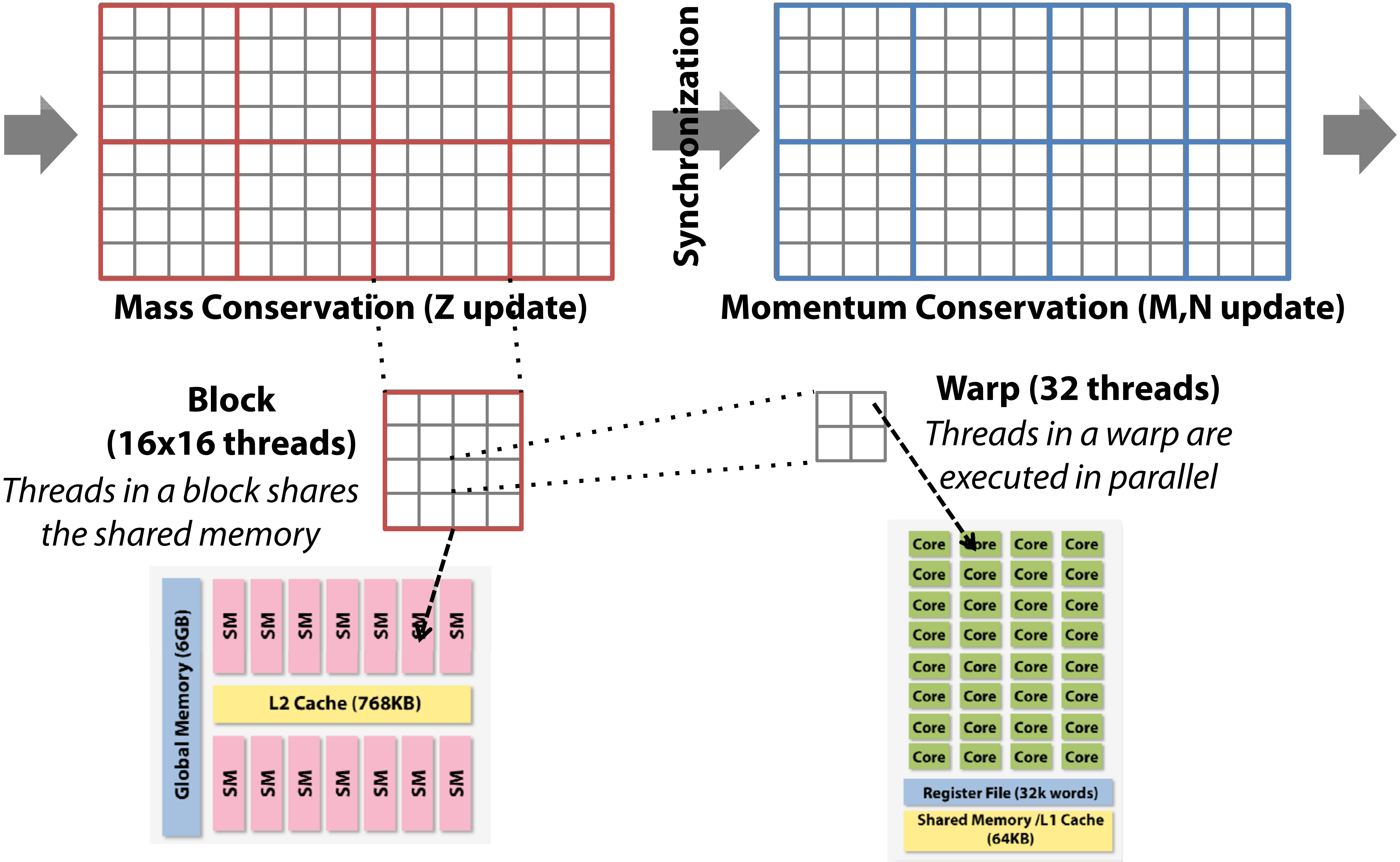


NVIDIA Tesla C2075
(Fermi architecture)
14 Streaming Multiprocessors
6GB Main Memory
768KB L2 Cache

Streaming Multiprocessor (SM)
32 Integer & FP cores

Naïve GPGPU Implementation

[Gidra et al., IEEE HPCC 2011]



Synchronization

Mass Conservation (Z update)

Momentum Conservation (M,N update)

Block
(16x16 threads)
Threads in a block shares the shared memory

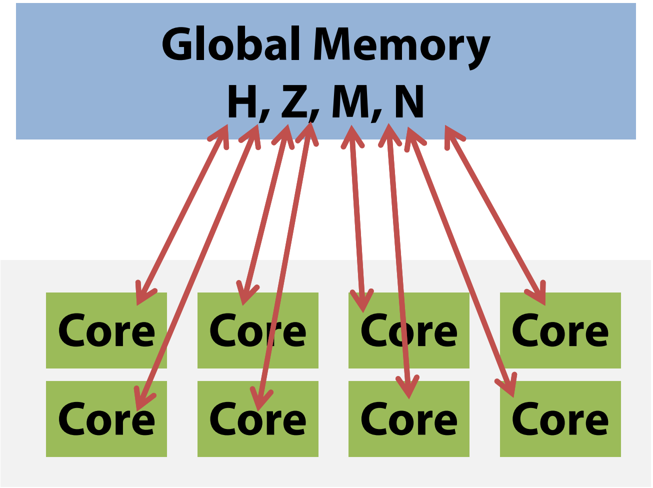
Warp (32 threads)
Threads in a warp are executed in parallel

Performance Bottleneck

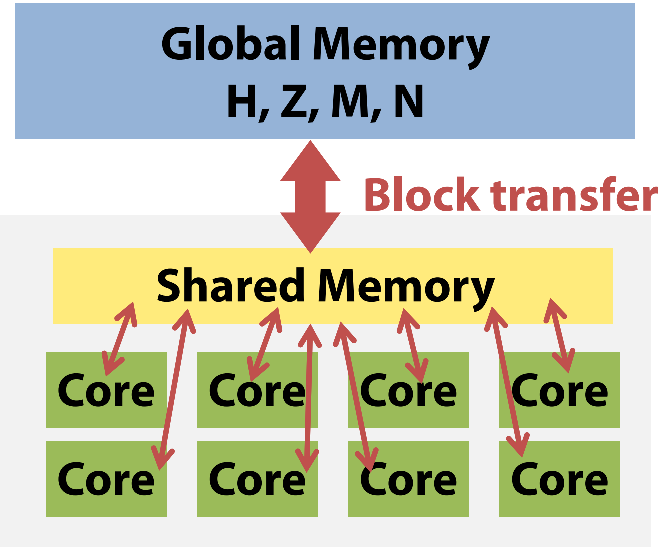
- Runtime is dominated by global memory accesses
  - #global accesses
    - Mass: Read H,Z,M,N (1040x668x6), Write Z (1040x668)
    - Momentum: Read H,Z,M,N (1040x668x6), Write M,N (1040x668x2)
    - Total: 1040x668x12 reads & 1040x668x3 writes
  - Global memory synchronization between Mass and Momentum
- How to reduce the accesses?
  - **Technique 1:** Using shared memory to share H,Z,M,N between Mass and Momentum
    - Can eliminate all H,Z,M,N read in Momentum
  - **Technique 2:** Merging Mass and Momentum to eliminate global memory synchronization
    - More chance to utilize computation cores during memory access

Technique 1: Using Shared Memory

- For each block, (H,Z,M,N) are loaded to shared memory
  - #global accesses
    - Mass: Read H,Z,M,N (1040x668x**4**), Write Z (1040x668)
    - Momentum: Write M,N (1040x668x2)
    - Total: 1040x668x4 reads (67% reduction) & 1040x668x3 writes



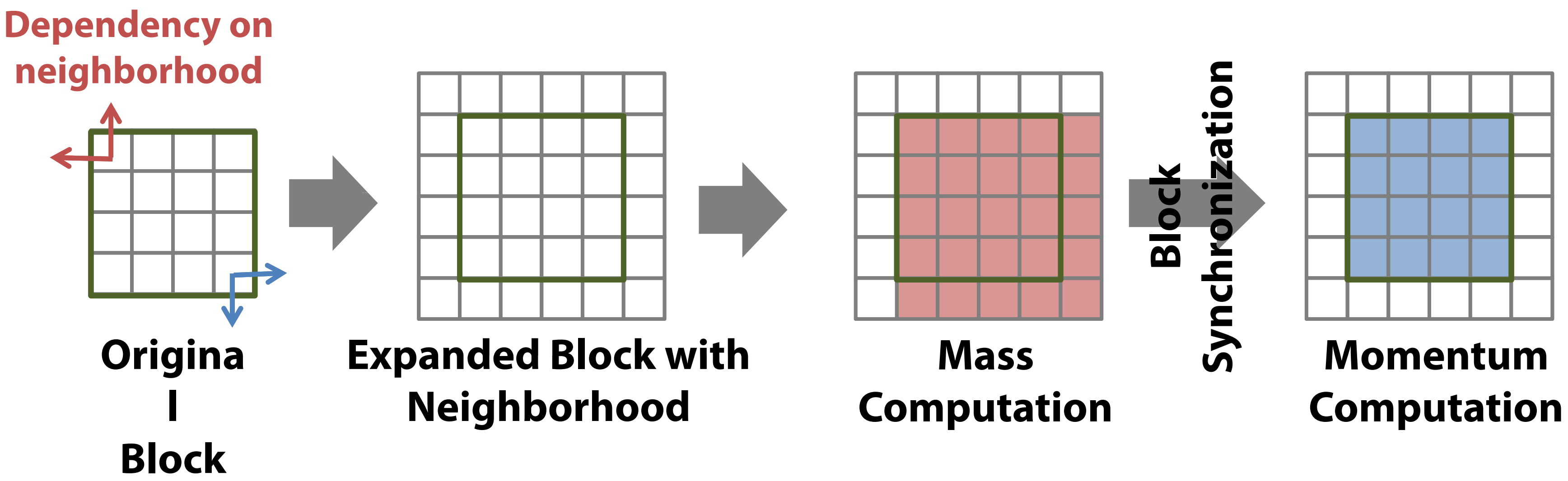
Original Implementation



Shared Memory Implementation

Technique 2: Eliminating Synchronization

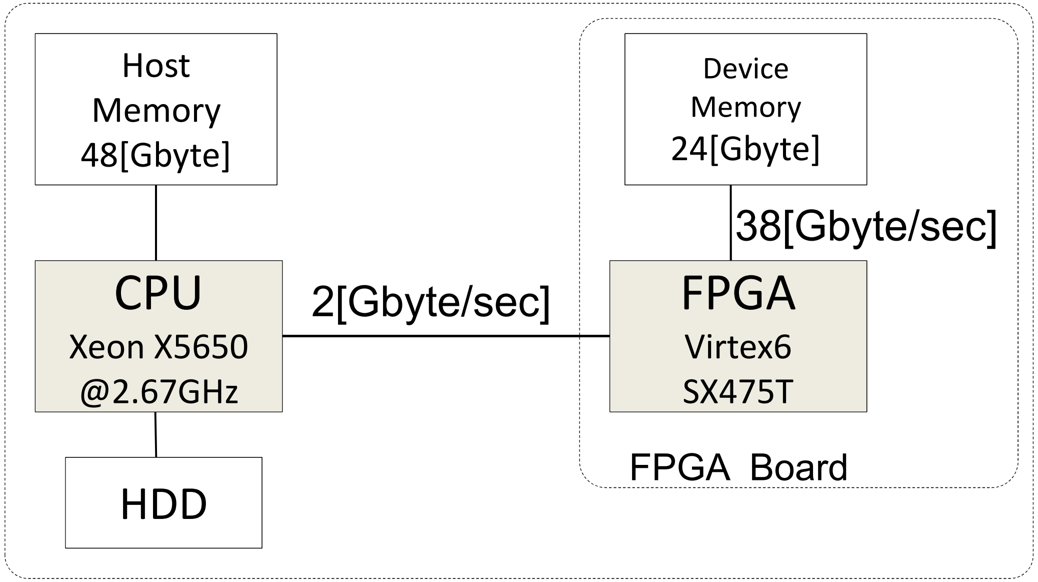
- Global synchronization can be eliminated by merging Mass and Momentum functions
  - However, Mass and Momentum depend on neighboring values of the block
    - Neighboring values are loaded onto the shared memory
    - Neighboring Z values are also computed
    - Duplicated load & computation do not impact on runtime



Experimental Results

- CUDA based implementation
- Runtime of 7200 iterations (2 hours)
- Original C implementation
  - Runtime: 78.7 seconds
- Naïve GPGPU implementation
  - Runtime: 2.75 seconds (28.6X speedup)
- Our GPGPU implementation
  - Runtime: 1.96 seconds (40.2X speedup)

Overview of FPGA System



Host Memory 48[Gbyte]

CPU

Xeon X5650 @2.67GHz

HDD

2[Gbyte/sec]

Device Memory 24[Gbyte]

38[Gbyte/sec]

FPGA

Virtex6 SX475T

FPGA Board

FPGA(Virtex6 SX475T) Resources

LUT	297600
FF	595200
BRAM	1064
DSP	2016

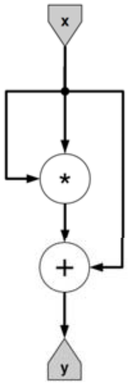
Host Code (C)

```
int in_data[n] = {1,2,3,4,5};
int out_data[n];

run_fpga(
            input(“x”, in_data),
            output(“result”,out_data)
,
            run_cycle(“Example”,n)
)
```

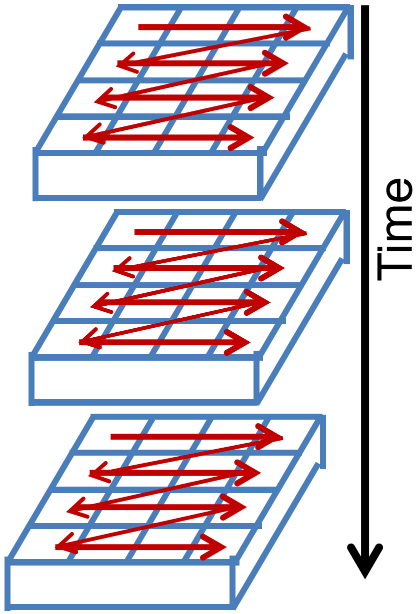
Data Flow Graph (MaxJava)

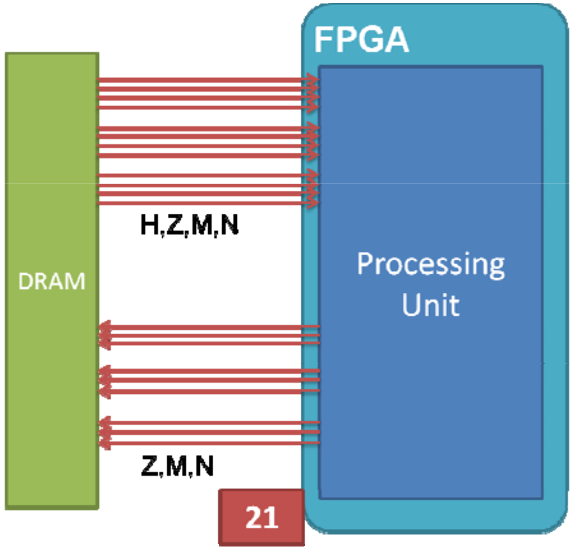
```
Public class Example{
            x = input();
            y = x*x + x;
            y = output()
}
```



Strategy for FPGA implementation

- As shown earlier, stream is based on each time step computation
  - Like to keep communication between FPGA and DRAM as small amount as possible

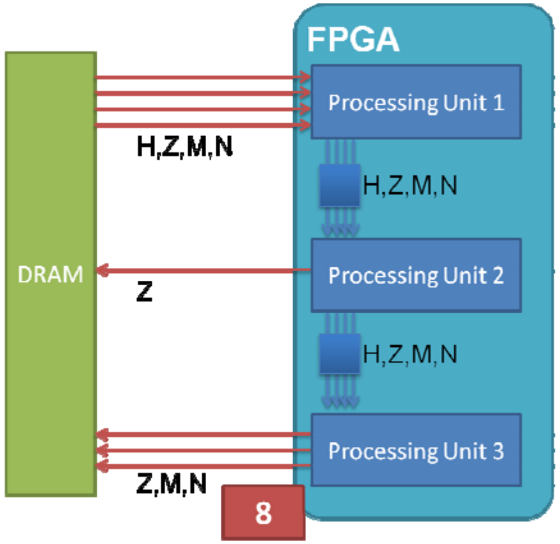




Stream for each region

- 84*4[Byte]*200[MHz] =**67**[GByte/s]

  for 12 time steps



Stream for each time step

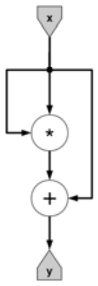
- 17*4[Byte]*200[MHz]= **13.6**[GByte/s]

  for 12 time steps

MaxCompiler

Data Flow Graph(MaxJava)

```
Public class Example{
            x = input();
            y = x*x + x;
            y = output()
}
```

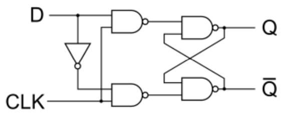


MaxCompiler

Automatic Pipelining according to the **DFG** and **Clock Frequency**

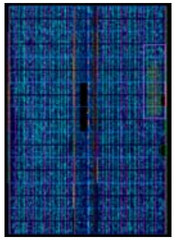
RTL

Logic Synthesis



Gate Level

Placing and Routing



FPGA Configuration

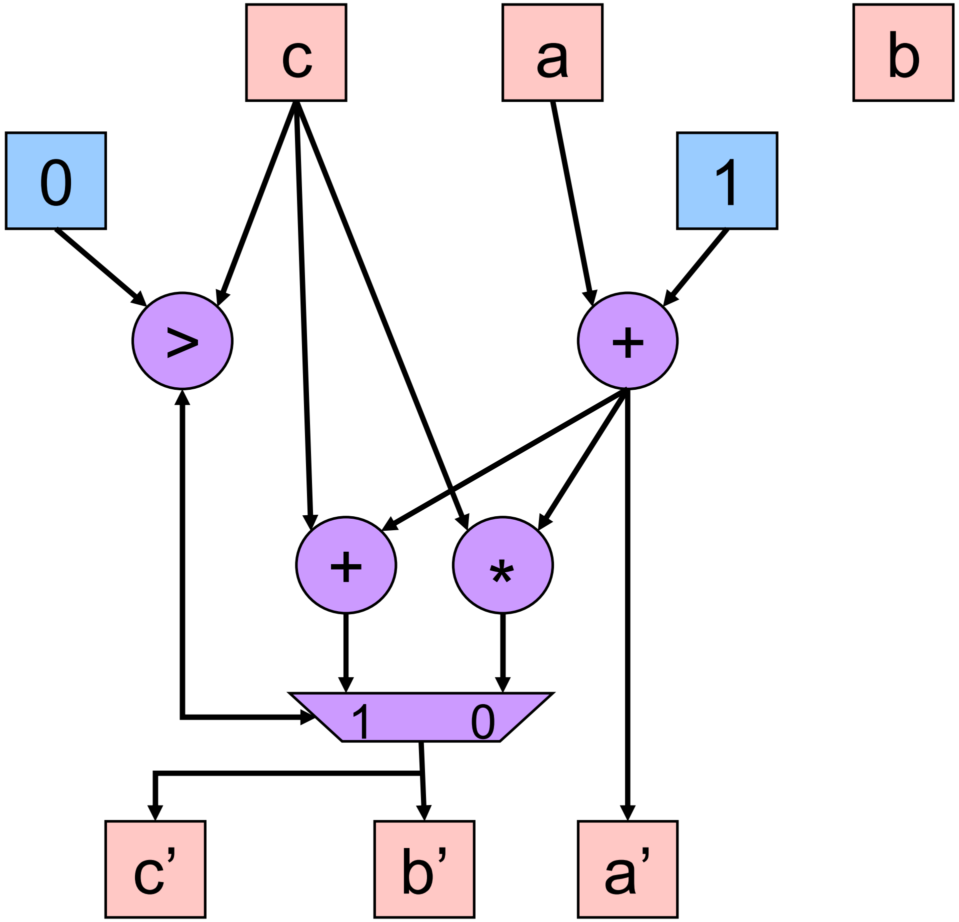
- Development using RTL require time and effort
- DFG is more abstract and reduces the development time
- This enable us to try more design alternatives

DFG example

```
int a, b, c;

void fct()
{
  a++;
  if (c > 0)
        b = a + c;
  else
        b = a * c;
  c = b;
}
```

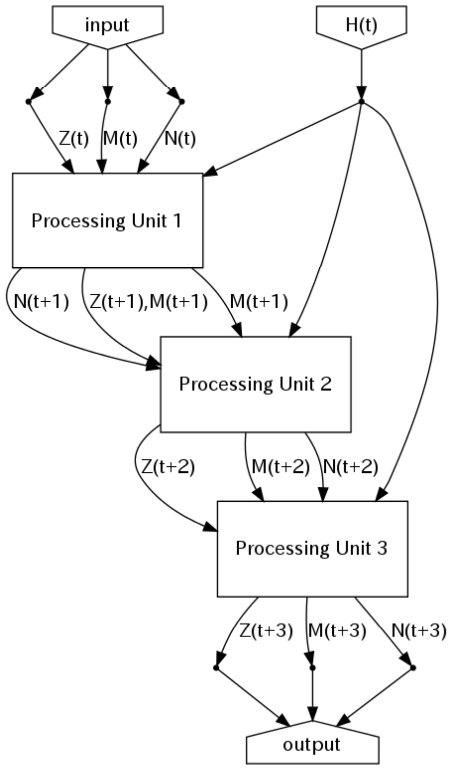
Generate DFG corresponding to as large as possible portions of codes



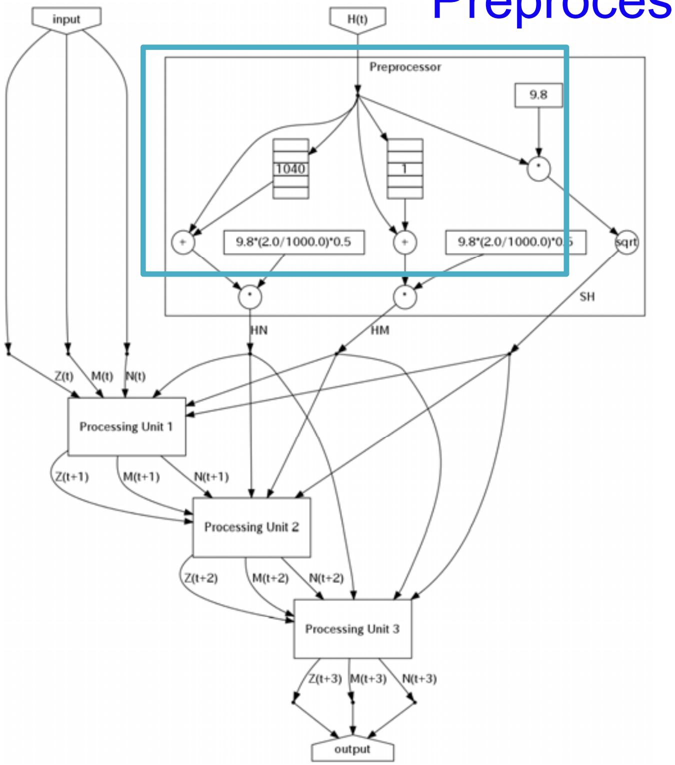
Optimizations

- Final Implementation has over 1,200 pipeline stages

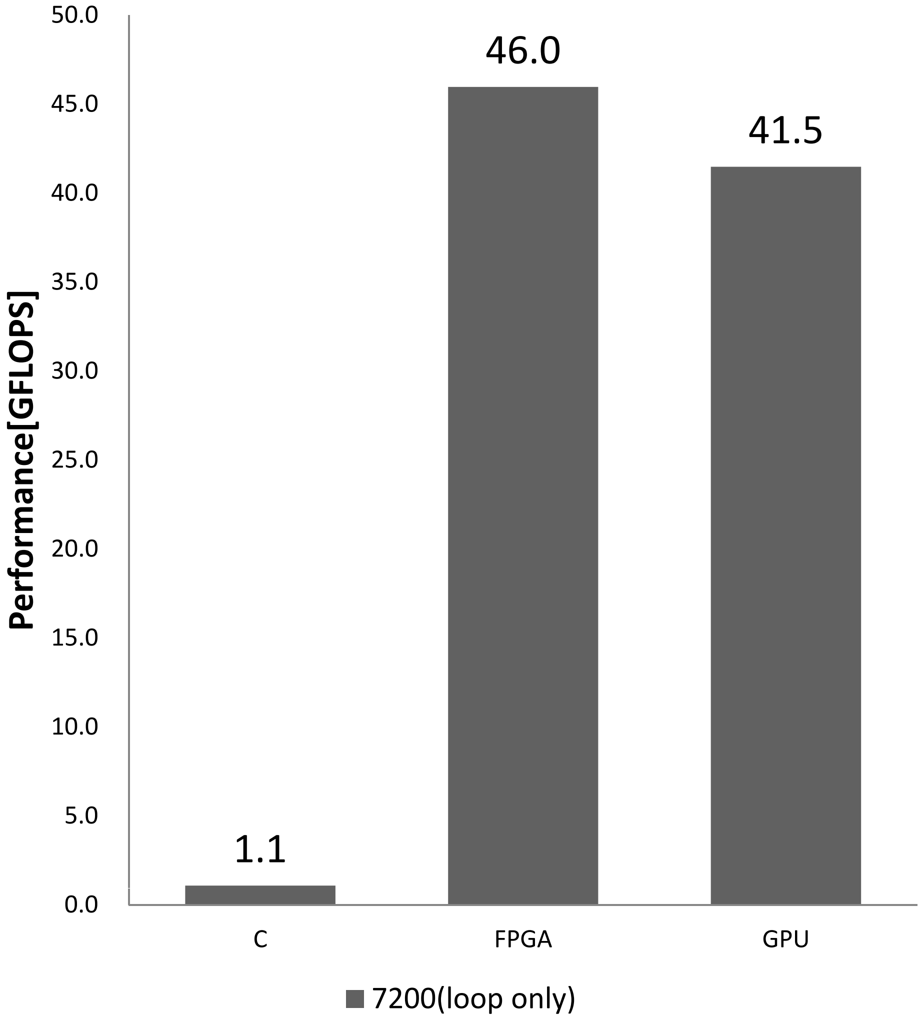
Before



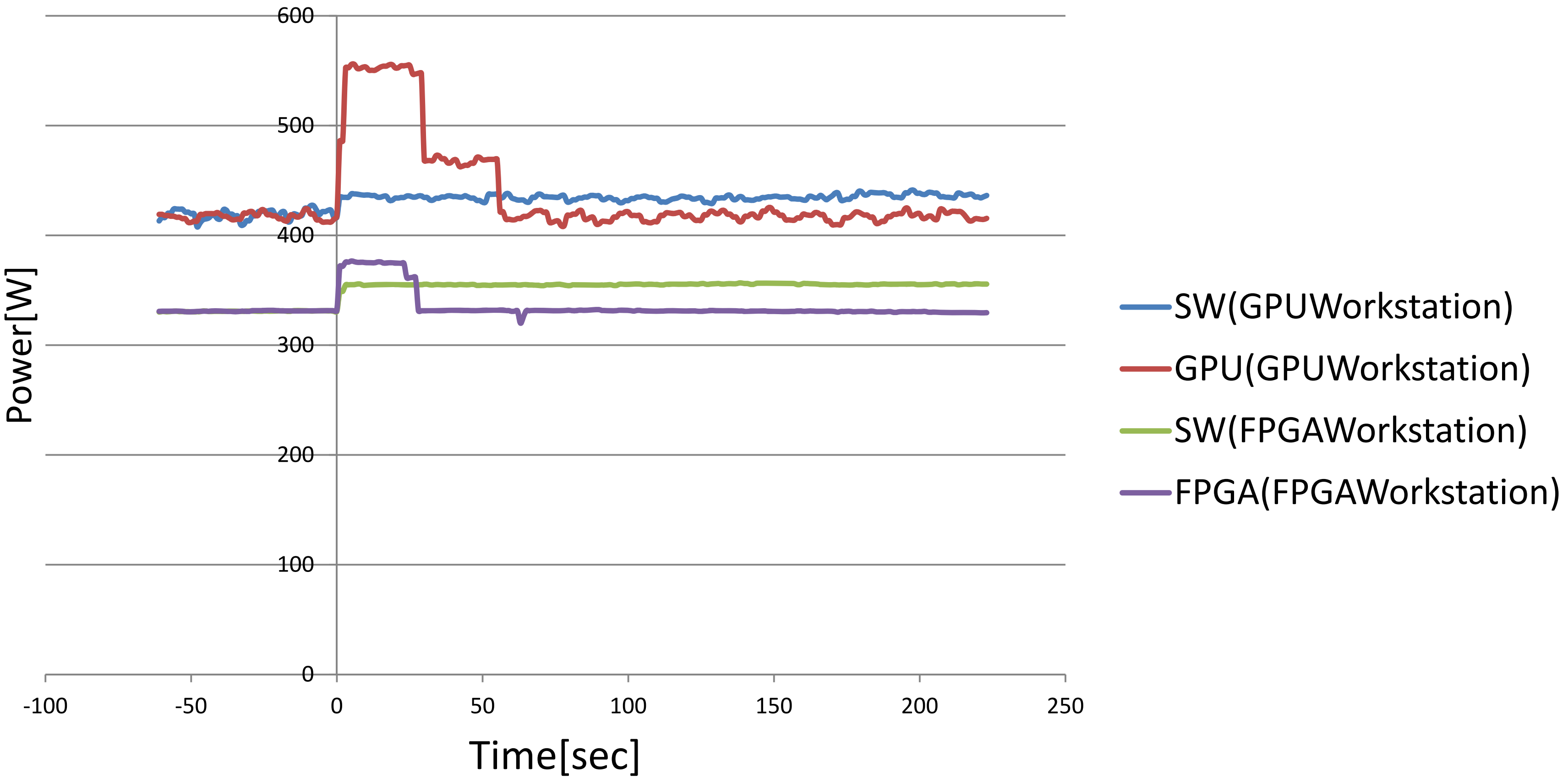
After


Preprocess Unit

Performance of the main loop part

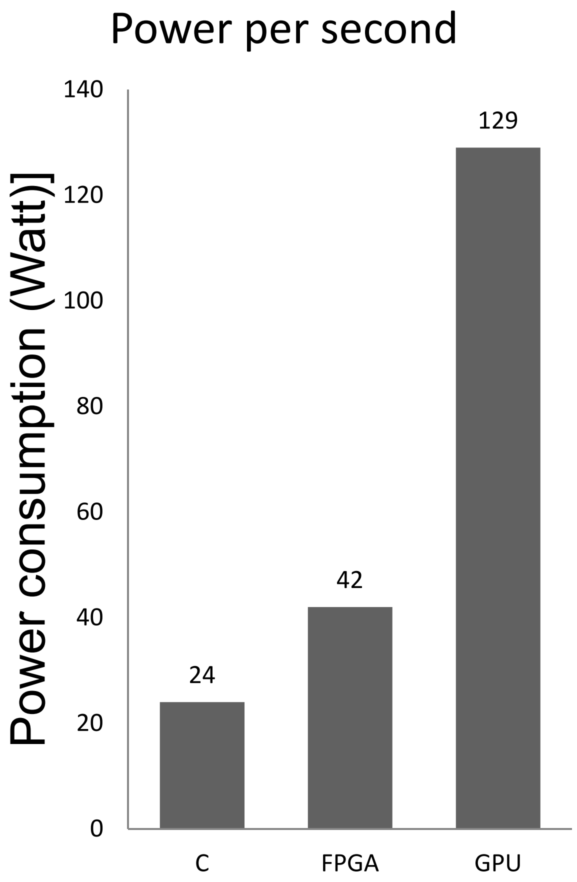


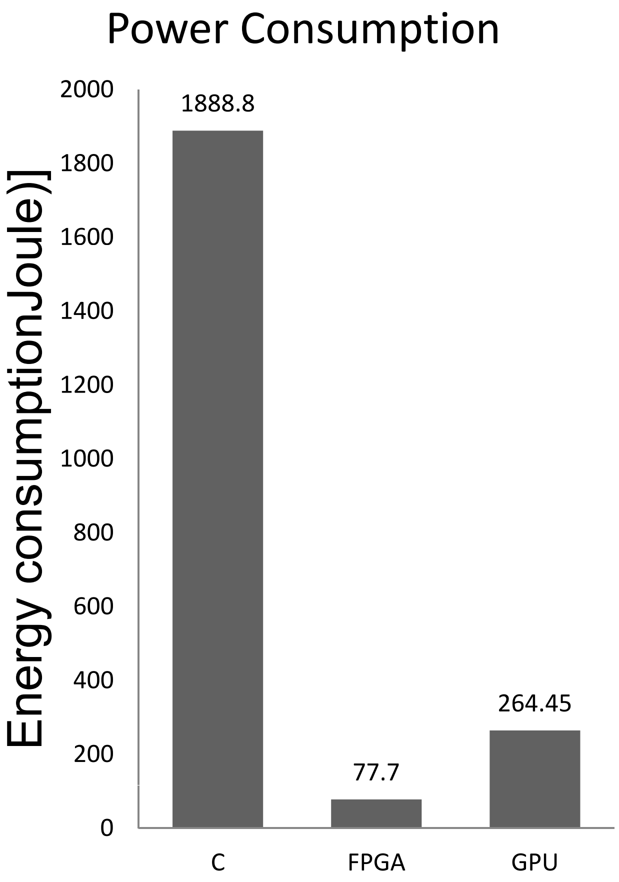
Power Consumption



Comparison of Power Consumption

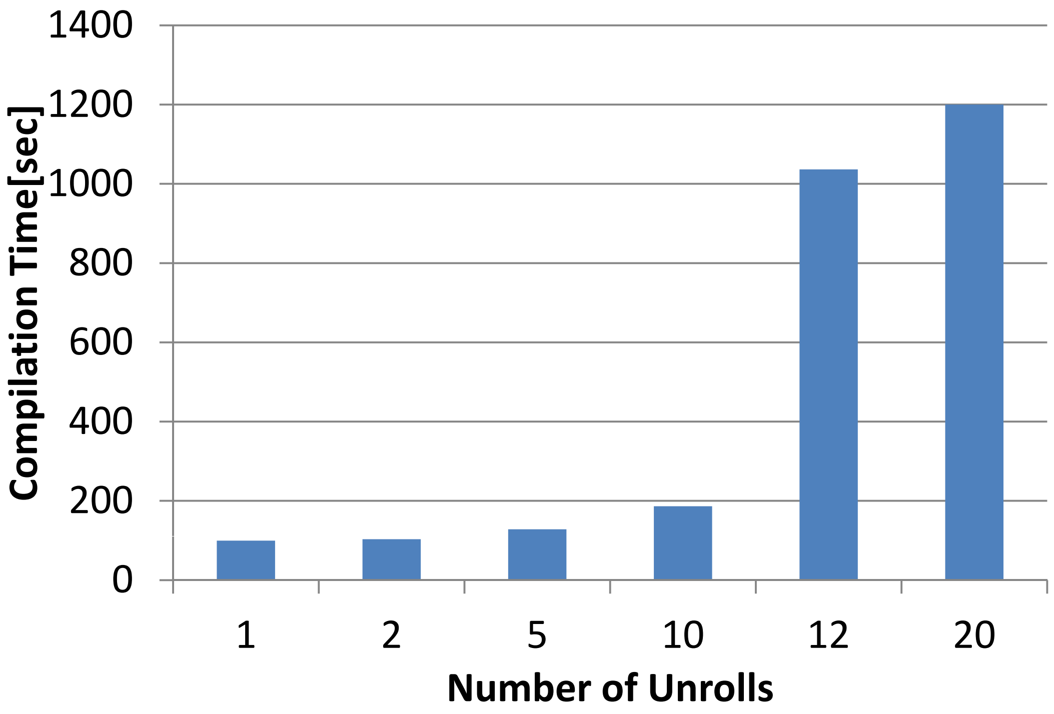
- FPGA is much better in terms of energy consumption





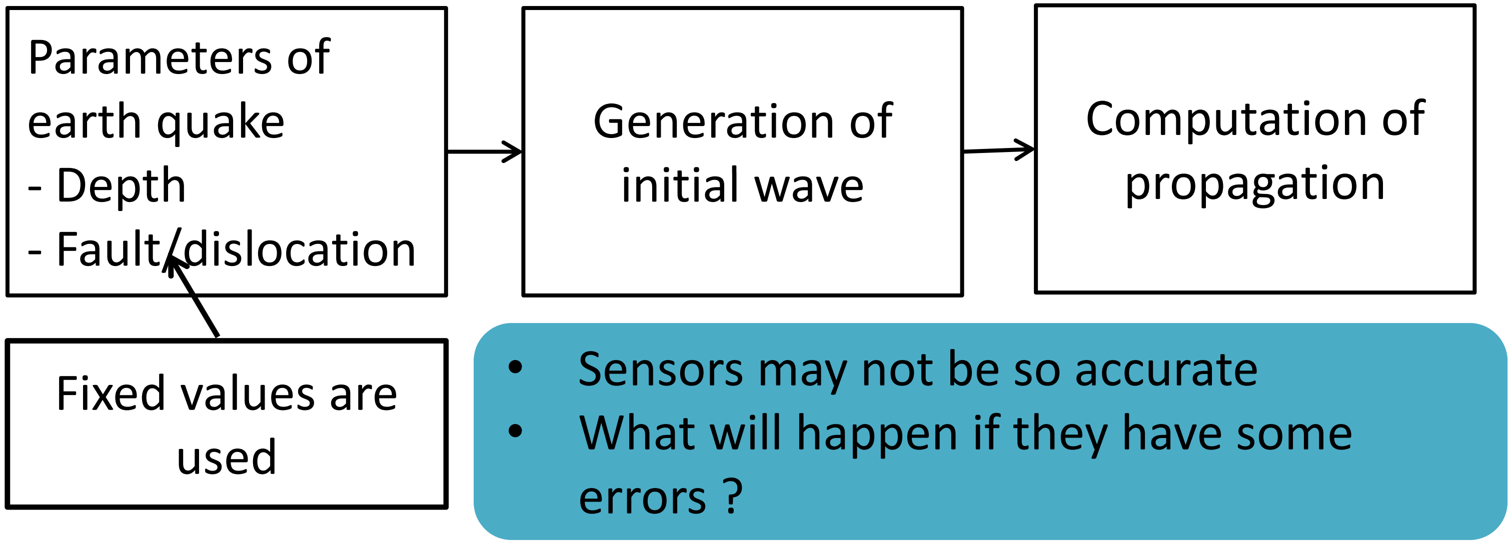
Compilation time

- Time to compile DFG into FPGA implementation
  - High/logic synthesis, placement & routing
- The relationship of the number of unrolls and compilation time



Statistical model checking on Tsunami simulation results

- Used the SMC developed by Prof. Clarke’s group
  - With Bayes statistics analysis
  - Software based



Parameters of earth quake
- Depth
- Fault/dislocation

Generation of initial wave

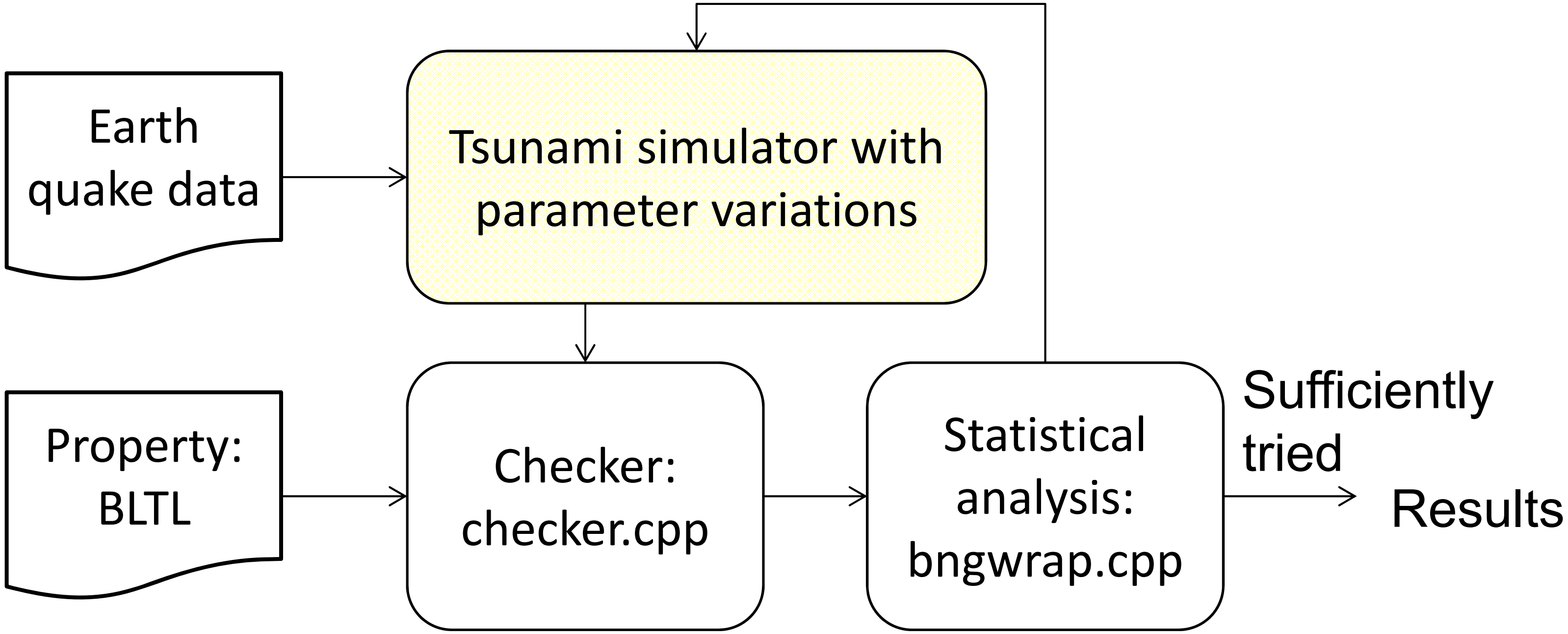
Computation of propagation

Fixed values are used

- Sensors may not be so accurate
- What will happen if they have some errors ?

Software implementation

- Used the SMC developed by Prof. Clarke's group
  - Only colored (yellow) ones are replace with ours



Results (1)

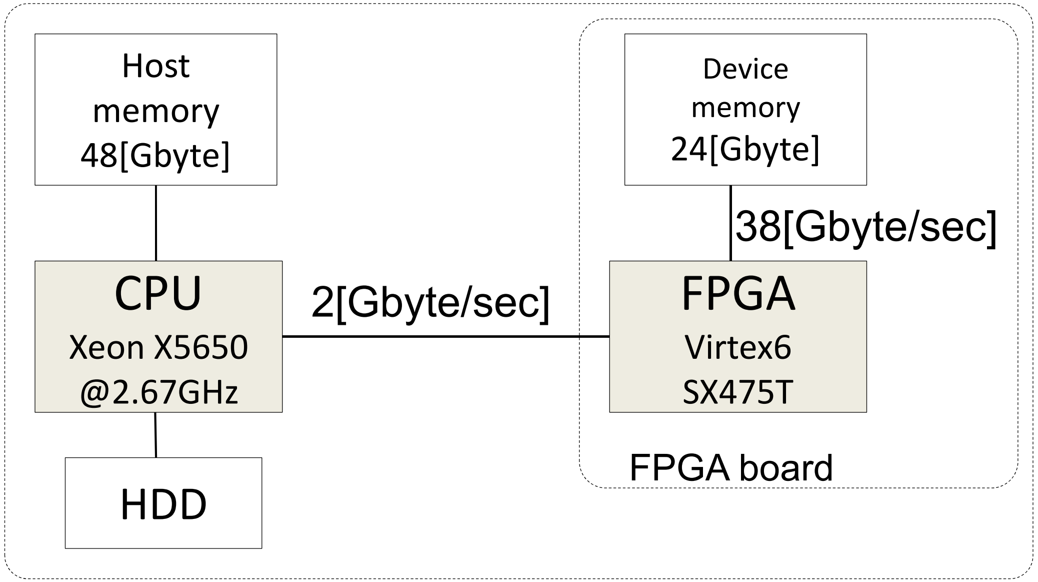
Parameters	Test	Property	A/R	Satisfy	All	Time[sec]
H, sigma=1%	BFT, 0.9, 1000, 1, 1	G[1800] (Z1 < 3.3)	R	0	3	149
		G[1800] (Z1 < 3.4)	R	0	3	149
Earth quake depth		G[1800] (Z1 < 3.5)	A	44	44	1635
		G[1800] (Z1 < 3.6)	A	44	44	1635
		G[1800] (Z1 < 3.7)	A	44	44	1635
		G[1800] (Z1 < 3.8)	A	44	44	1635
H, sigma=1%	BFT, 0.99, 1000, 1, 1	G[1800] (Z1 < 3.3)	R	0	2	74
		G[1800] (Z1 < 3.4)	R	0	2	74
Earth quake depth		G[1800] (Z1 < 3.5)	A	239	239	8962
		G[1800] (Z1 < 3.6)	A	239	239	8962
		G[1800] (Z1 < 3.7)	A	239	239	8962
		G[1800] (Z1 < 3.8)	A	239	239	8962
L, W, sigma=5%	BFT, 0.9, 1000, 1, 1	G[1800] (Z1 < 3.3)	R	0	3	149
		G[1800] (Z1 < 3.4)	R	0	3	149
Fault/dislocation		G[1800] (Z1 < 3.5)	A	224	237	8865
length and width		G[1800] (Z1 < 3.6)	A	44	44	1638
		G[1800] (Z1 < 3.7)	A	44	44	1638
		G[1800] (Z1 < 3.8)	A	44	44	1638
L, W, sigma=5%	BFT, 0.99, 1000, 1, 1	G[1800] (Z1 < 3.3)	R	0	2	77
		G[1800] (Z1 < 3.4)	R	4	7	299
Fault/dislocation		G[1800] (Z1 < 3.5)	R	306	319	11927
length and width		G[1800] (Z1 < 3.6)	A	239	239	8939
		G[1800] (Z1 < 3.7)	A	239	239	8939
		G[1800] (Z1 < 3.8)	A	239	239	8939

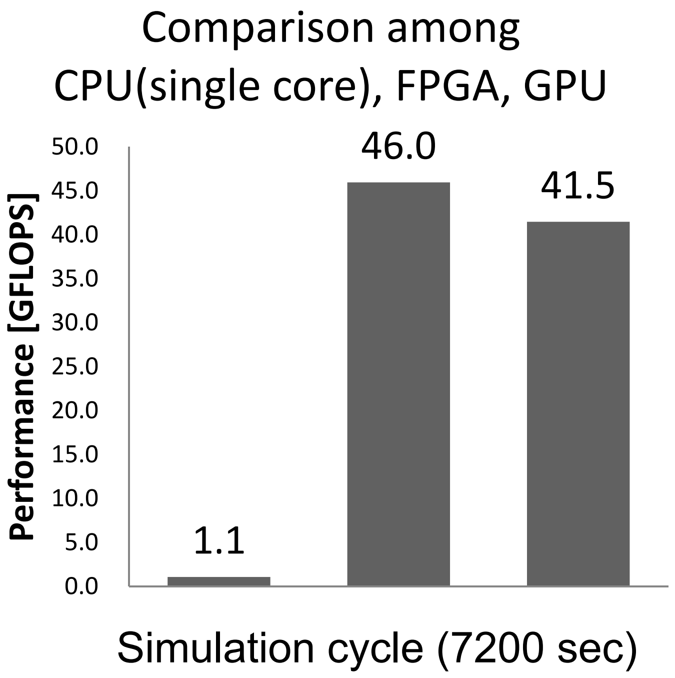
Results (2)

Parameters	Test	Property	p	Satisfy	All	Time[sec]
H, sigma=1%	BEST,0.05,0.9,1,1	G[1800] (Z1 ＜ 3.3)	0.0434783	0	21	817
	(C−H Bound : 460)	G[1800] (Z1 ＜ 3.4)	0.0434783	0	21	817
		G[1800] (Z1 ＜ 3.5)	0.956522	21	21	817
		G[1800] (Z1 ＜ 3.6)	0.956522	21	21	817
		G[1800] (Z1 ＜ 3.7)	0.956522	21	21	817
		G[1800] (Z1 ＜ 3.8)	0.956522	21	21	817
L, W, sigma=5%	BEST,0.05,0.9,1,1	G[1800] (Z1 ＜ 3.3)	0.0434783	0	21	796
	(C−H Bound : 460)	G[1800] (Z1 ＜ 3.4)	0.430189	113	263	9531
		G[1800] (Z1 ＜ 3.5)	0.956522	21	21	796
		G[1800] (Z1 ＜ 3.6)	0.956522	21	21	796
		G[1800] (Z1 ＜ 3.7)	0.956522	21	21	796
		G[1800] (Z1 ＜ 3.8)	0.956522	21	21	796
L, W, sigma=5%	BEST, 0.01, 0.9, 1, 1	G[1800] (Z1 ＜ 3.3)	0.0251177	15	635	23803
	(C−H Bound : 11513)	G[1800] (Z1 ＜ 3.4)	0.424508	2805	6608	249805
		G[1800] (Z1 ＜ 3.5)	0.96146	947	984	36908
		G[1800] (Z1 ＜ 3.6)	0.991304	113	113	4229
		G[1800] (Z1 ＜ 3.7)	0.991304	113	113	4229
		G[1800] (Z1 ＜ 3.8)	0.991304	113	113	4229

Acceleration by HW Implementation

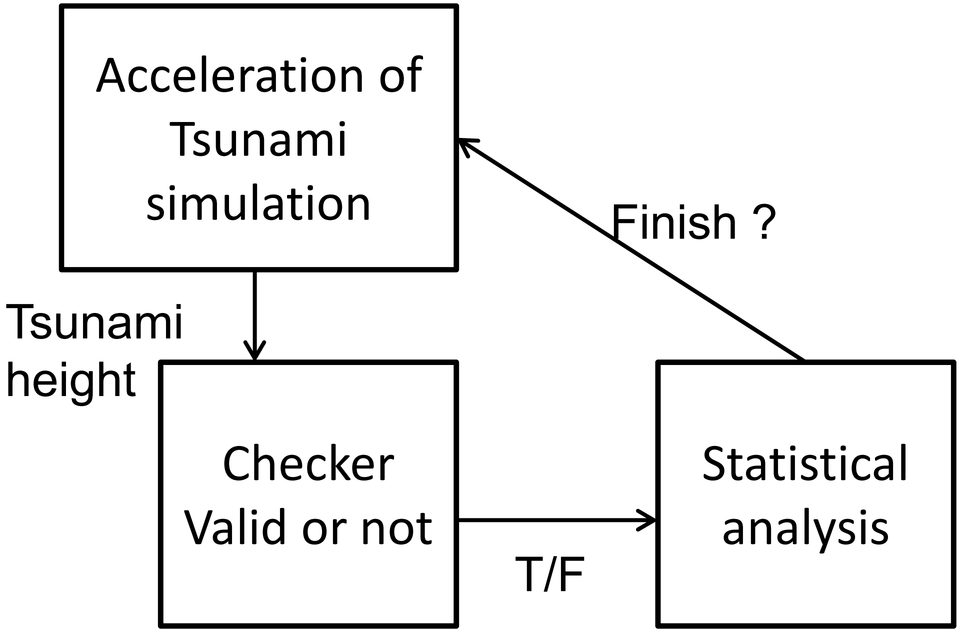
- Main loop of TUNAMI simulation can be 46.0 times faster
- In case of GPU, 41.5 time acceleration is realized (just for reference)




Comparison among CPU(single core), FPGA, GPU

How can we speed up statistical model checking

- Tsunami simulation can be accelerated with FPGA/GPU by 40 times or more
  - But data transfer speed between FPGA/GPU board and microprocessor (PCI-e) is not so fast

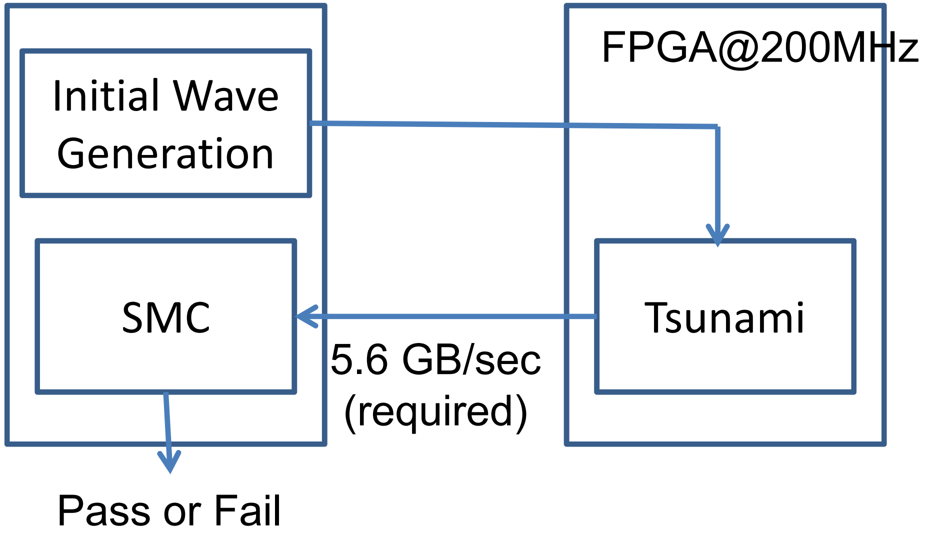


Data Transfer b/w Host and FPGA

- Results of Tsunami simulation should be transferred from FPGA/GPU to host processor
  - FPGA → Host: 2Gbyte/sec (by PCI Express bus)
- FPGA-based Tsunami simulation needs:
  - 28 byte data / clock cycle (16 byte for input, 12 byte for output)
  - Needs 5.6Gbyte/sec @ (200MHz FPGA)
- Considering data transfer, actual acceleration by FPGA-based implementation is 16 times

Statistical Model Checking of Tsunami Simulation Results

- SMC of Tsunami simulation can be accelerated by hardware implementation
  - Data transfer can be reduced
  - Can fully utilize the acceleration of Tsunami simulation in SMC



FPGA@200MHz

Initial Wave Generation

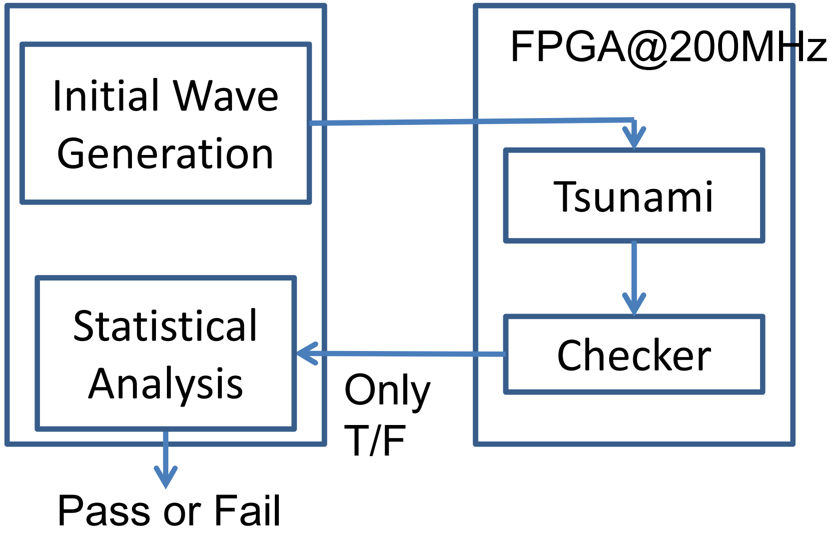
SMC

Tsunami

5.6 GB/sec (required)

Pass or Fail

SW implementation of SMC part



FPGA@200MHz

Initial Wave Generation

Tsunami

Checker

Statistical Analysis

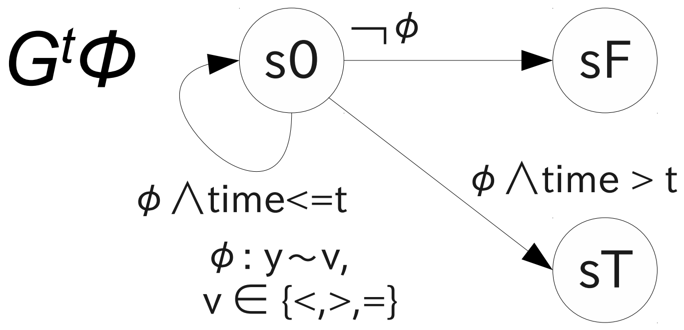
Only T/F

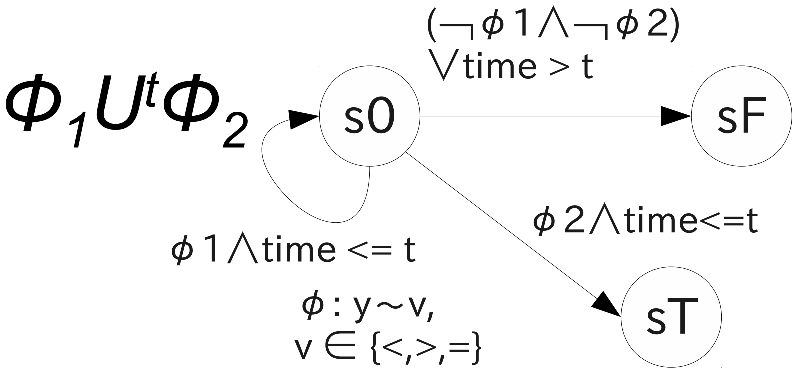
Pass or Fail

HW implementation of (partial) SMC part

HW Implementation of “checker”

- With FSM for each property

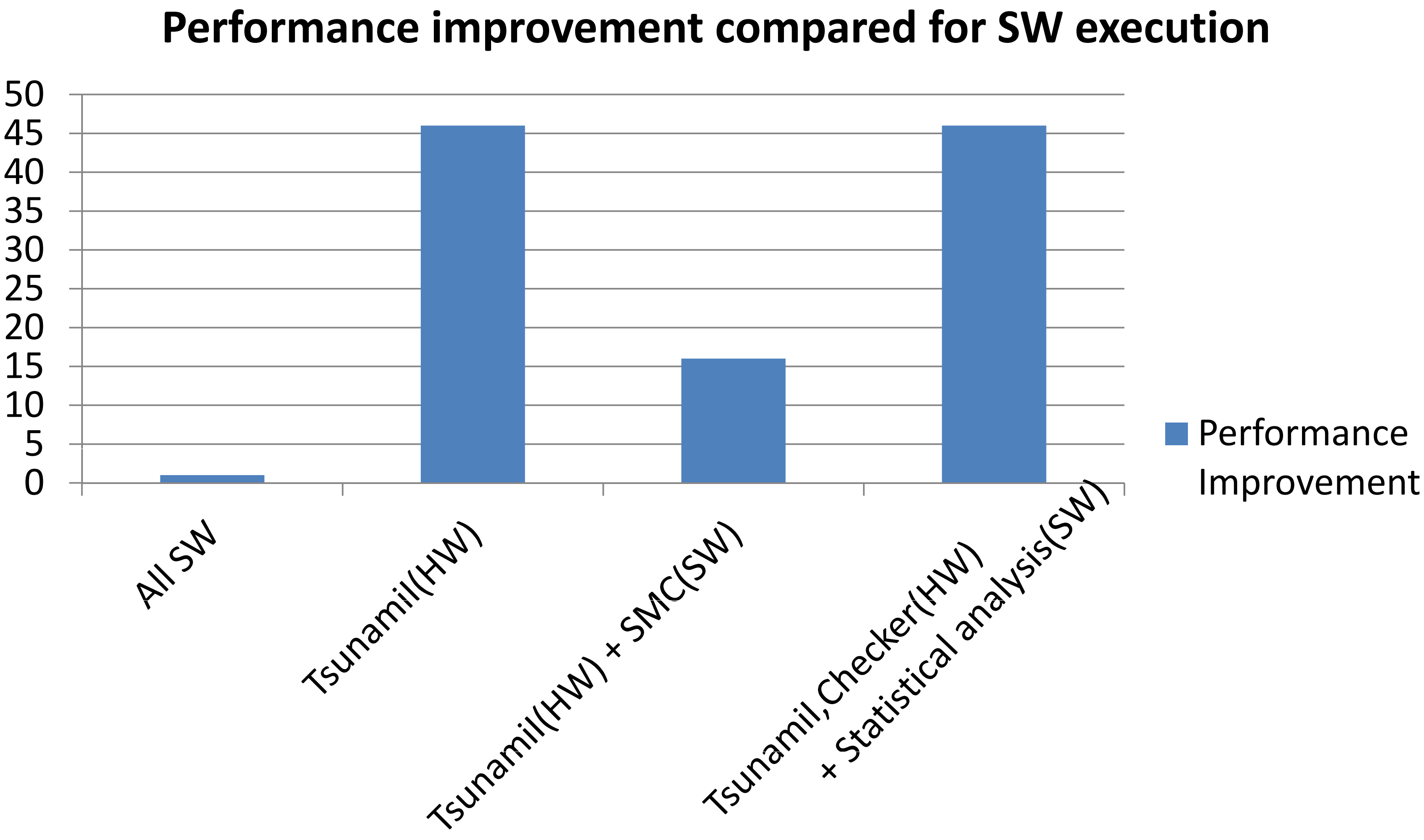




- With model checking of the traces
  - LTL formulae can be checked in a bottom up way with linear time
  - Example: F(a->(b U c))
    - Check each sub-formula
    - Combine in bottom up way

Time	1234567891111 0123
a	0101010111011
b	1110001001111
c	0011110001111
b U c	1111110001111
a->(b U c)	1111111001111
F(a->(b U c))	1111111111111

Performance Improvement of SMC of Tsunami Simulation



Conclusions and on-going works

- Tsunami simulation has been accelerated by 40-45 times
  - Space decomposition with GPU
  - Time-wise pipelining with FPGA
- Statistical model checking on Tsunami simulation results
  - Could be time consuming with SW only implementation (15 X speed up)
  - By HW implementation of checker, 40X achieved
- Entire HW implementation is on-going
  - Target example: Rounding robustness of floating point computation with Monte Carlo Arithmetic