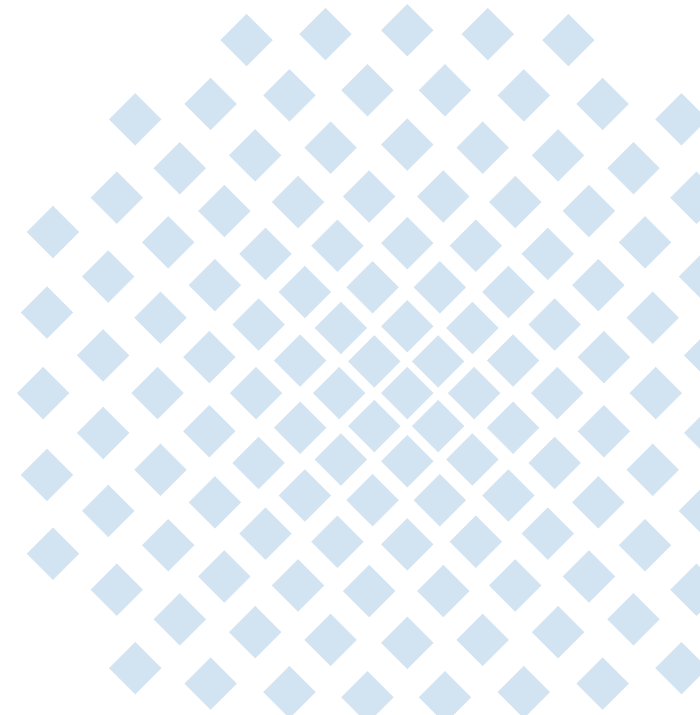


Simulation tools and realization approaches using dedicated hardware for accelerating radio channel computation

ITG 5.2.4 Workshop "Simulating Mobile Networks"

Matthias Kaschub
matthias.kaschub@ikr.uni-stuttgart.de
Stuttgart, 19.06.2008

Universität Stuttgart
Institute of Communication Networks
and Computer Engineering (IKR)
Prof. Dr.-Ing. Dr. h.c. mult. P. J. Kühn



Outline

- Motivation
- Current situation
- Parallelism
 - Overview
 - Types of parallelism
 - Instruction level
 - Fine grained
 - Task parallelism
 - Batch parallelism
 - Job offloading
- Hardware architectures
 - IBM Cell processor
 - NVIDIA Cuda
- Conclusion

Motivation

Premise

Computational complexity of simulation increases

- Systems under examination becoming more complex
 - Systems incorporate more and more algorithms and mechanisms
ARQ, HARQ, CQI, MIMO, ICIC, FSPS
 - Feedback: everything is influencing everything else
No layering in scheduling anymore
- Multiscale models
 - Algorithms work on multiple timescales
 - Algorithms and its effect on different timescales
- Computational complex algorithms
 - Complex algorithms in standards and real systems
Examples: FFT, matrix inversion, optimization problems ...
 - Research: more general solutions
 - Before developing heuristics and for benchmarking heuristics
 - Examples: genetic algorithms, LP, graph theory...

Motivation

Profiling

Main performance killers in current simulations of mobile access networks

- Message passing
 - Classical event driven simulation
- Physical models
 - Channel models
 - User behavior
 - Raytracing
- Algorithms
 - Optimization: LP, ILP, GA, SA...
 - Graph algorithms: routing, coloring
 - Machine learning: classification, pattern recognition
 - Signal processing

Motivation

Why parallelization

Why not just run simulation with different parameters in parallel?

- Simulation time will increase:
 - Simulation programs keep getting more complex
 - Speed of single CPU core is not going to increase much anymore
- Simulation time >12h (over night) bad for productivity
- Debugging
- Bachelor thesis is only 3 or 4 months

Current situation

Properties of event-driven network simulation

- Single threaded
 - Single "calendar"
 - Events processed strictly sequentially
- High connectivity
 - Every object may (indirectly) access every other object
- Unclear bottlenecks
 - Usually no method that executes for $> 1s$
- No (external) blocking

Outline

- Motivation
- Current situation
- Parallelism
 - Overview
 - Types of parallelism
 - Instruction level
 - Fine grained
 - Task parallelism
 - Batch parallelism
 - Job offloading
- Hardware architectures
 - IBM Cell processor
 - NVIDIA Cuda
- Conclusion

Parallelism

Overview

Types of parallelism available in current computer systems

- Data & Instruction parallelism
 - Technology: AltiVec/SSE
- Task parallelism
 - Technology: OpenMP, Threads, processes
 - Variants:
 - Fine grained (loop parallelism; OpenMP)
 - Vertical split (pipelining on modeling level)
 - Horizontal split (parallelizing modules)
- Job offloading
 - Specialized hardware: NVIDIA Cuda, IBM Cell processor
- Batch parallelization
 - Simulation control

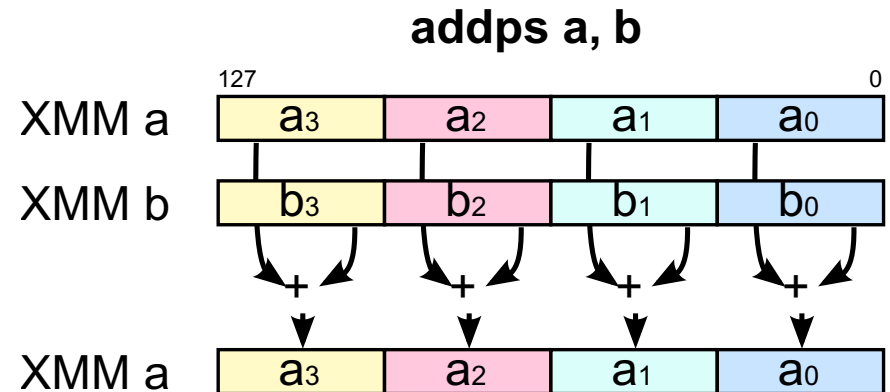
Parallelism

Data & instruction parallelism

Basic concept

- SIMD (Single Instruction Multiple Data)
- Available implementation
 - SSE (in modern x86 CPUs)
 - AltiVec (in modern PowerPC CPUs)
- Performance gain
 - Factor 2 ... 4 for single core
 - Data needs to be vectorizable
 - Mainly for number crunching

- **Instruction level**
- Fine grained
- Task parallelism
- Batch parallelism
- Job offloading



Problems

- Limited compiler support (except for special standard cases)
 - Hand-optimized code necessary (assembler knowledge)
- No big speedup in future processors expected

→ Suitable for problems that are easy to vectorize, limited effort, limited gain

Parallelism

Fine grained task parallelism (1)

Basic Idea

- Parallelizing event processing
- Using threads manually or with OpenMP
- Locking data structures

Problems

- High connectivity of code
 - Locking become performance bottleneck
- Such programs are hard to debug

- Instruction level
- **Fine grained**
- Task parallelism
- Batch parallelism
- Job offloading

→ That will never work

Parallelism

Fine grained task parallelism (2)

Basic idea

Different simulation paradigm:

Asynchronous messages instead of synchronous method calls

Properties

- Allows adding transparent fine grained parallelism
- Differences to traditional event-driven paradigm
 - Events are asynchronous messages
 - Each message handler publishes time cap (horizon)
 - Calendar schedules messages depending on timestamps and horizon

Drawbacks

- Lack of proper toolkits (user-friendly, transparent parallelism, low overhead)
- "Erlang" not widespread

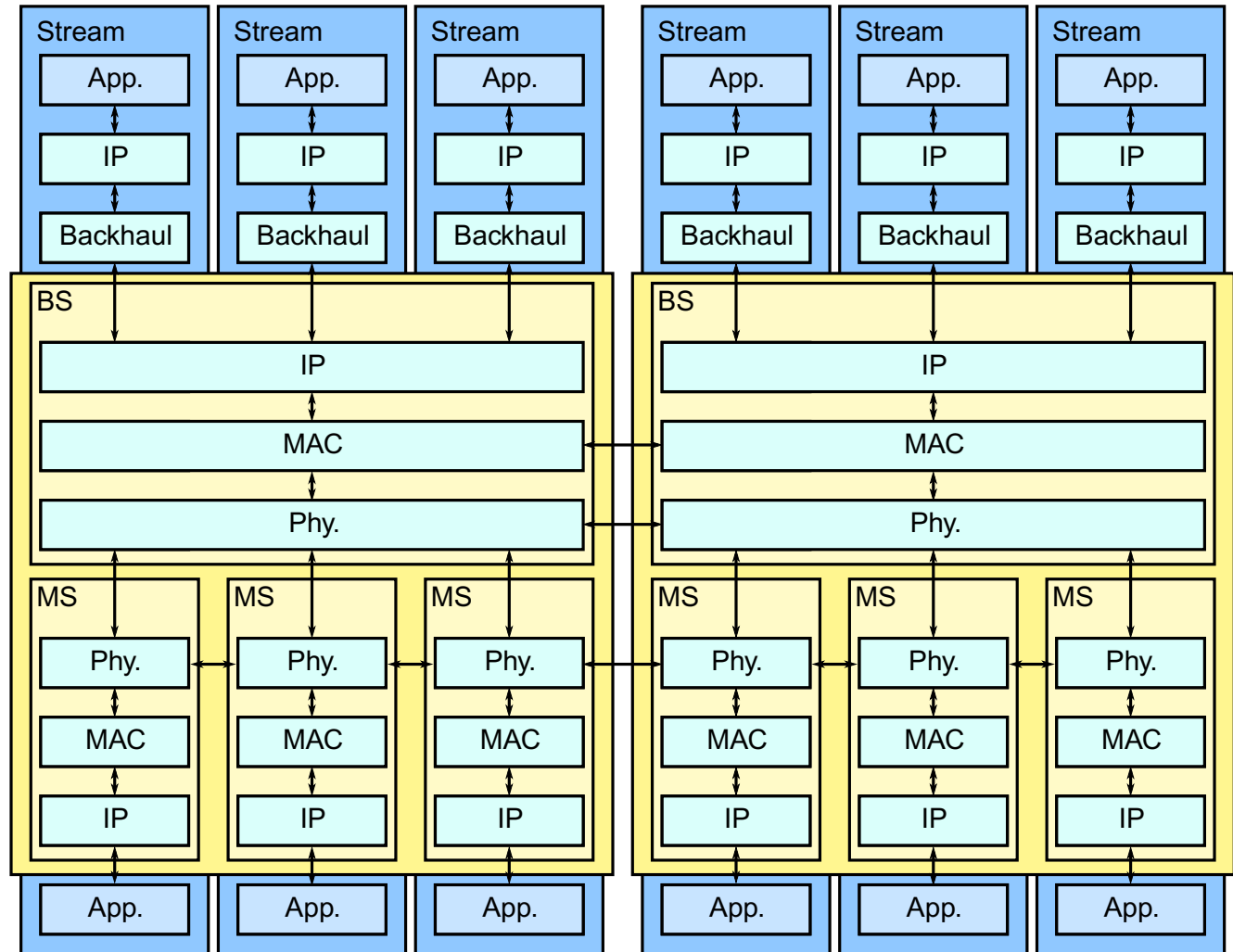
→ New paradigm, requires complete rewrite of existing code

- Instruction level
- **Fine grained**
- Task parallelism
- Batch parallelism
- Job offloading

Parallelism

Task parallelism

Typical simulation model:



Parallelism

Horizontal task parallelism

Basic idea

- Split problem into multiple layers
- Typical command line example:

```
./SimTraffic | SimMobility | \  
./SimChannels | ./SimMAC | \  
./SimEval
```

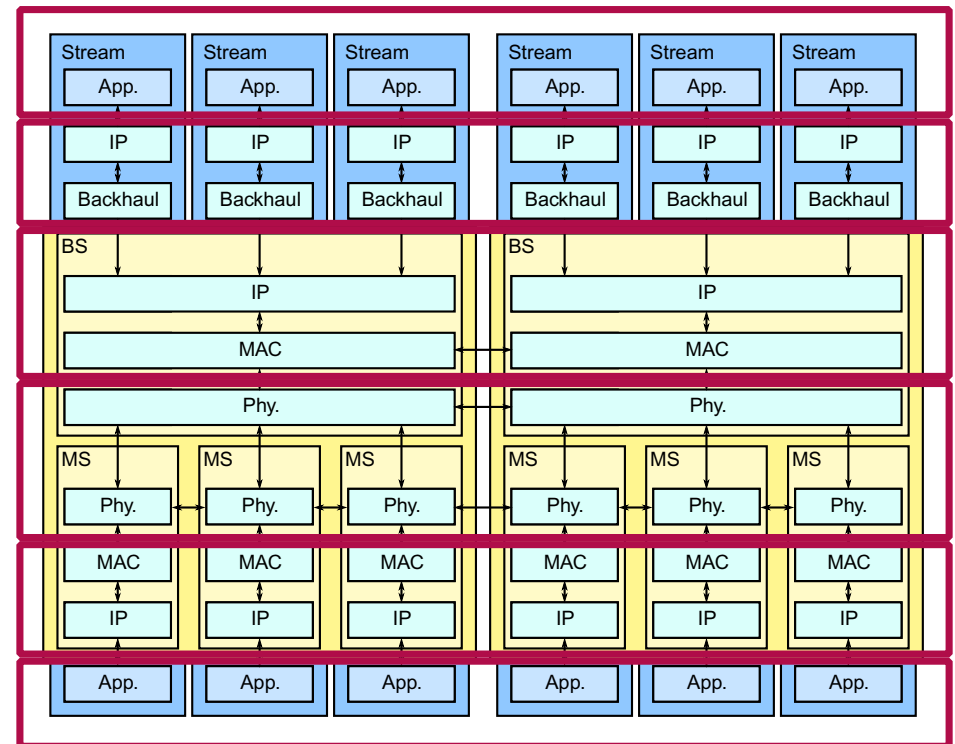
Advantages

- Inherent modularization
- Easier debugging
- Less resource conflicts (files ...)

Problems

- Parallelization limited to number of layers
- **Feedback** (bidirectional communication)

- Instruction level
- Fine grained
- **Task parallelism**
- Batch parallelism
- Job offloading



→ Suitable if the simulation model is unidirectional

Parallelism

Vertical task parallelism

Basic idea

- Split problem into blocks of same type
- e.g. one process/thread per mobile or per sector

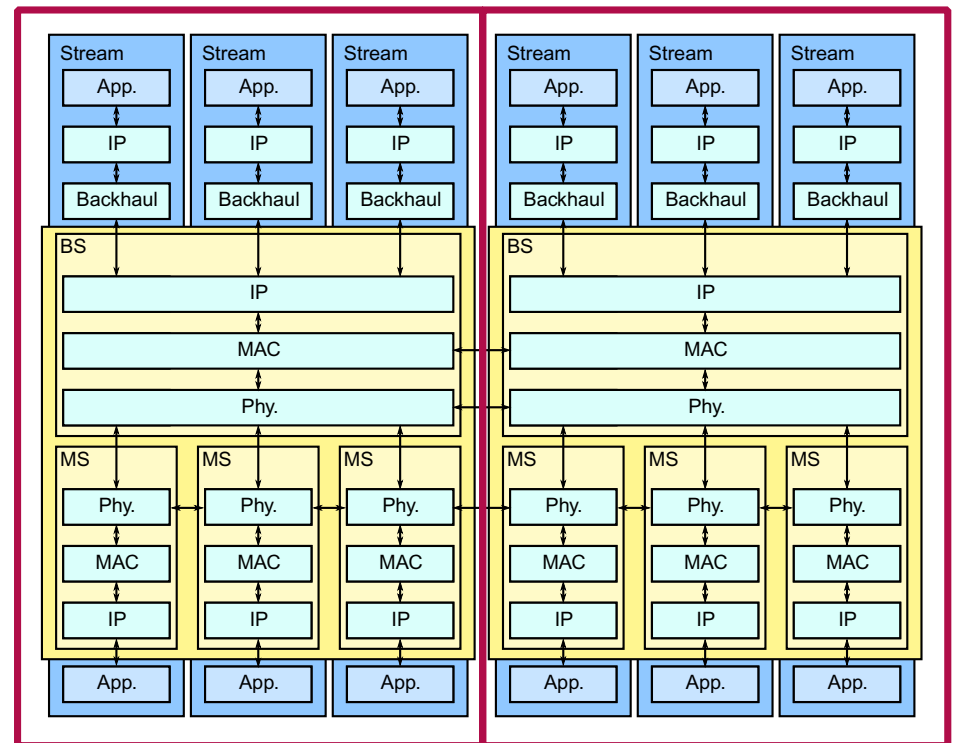
Advantages

- High degree of parallelization possible
- Clean approach

Problems

- Resource conflicts
- Hard to debug
- Communication and synchronization becomes bottleneck

- Instruction level
- Fine grained
- **Task parallelism**
- Batch parallelism
- Job offloading



→ Suitable only if model consists of nearly autonomous blocks (not for mobile networks)

Parallelism

Communication for task parallelism

- Shared memory (threads & mutexes)
 - Limited to one computer
 - Local IPC (SysV, Sockets)
 - Limited to one computer
 - Slow
 - Network protocols (RPC, RMI ...)
 - Large latency, overhead and complexity
 - Pipes, named pipes
 - Simple, but limited to one-way communication
 - Network transparency with "netcat"
 - Files
 - Simple, but limited to one-way communication
 - Very slow
 - Allows delayed processing (precalculation)
- Instruction level
 - Fine grained
 - **Task parallelism**
 - Batch parallelism
 - Job offloading

Parallelism

Batch parallelism

Basic idea

- Run multiple batches in parallel
- Aggregate results

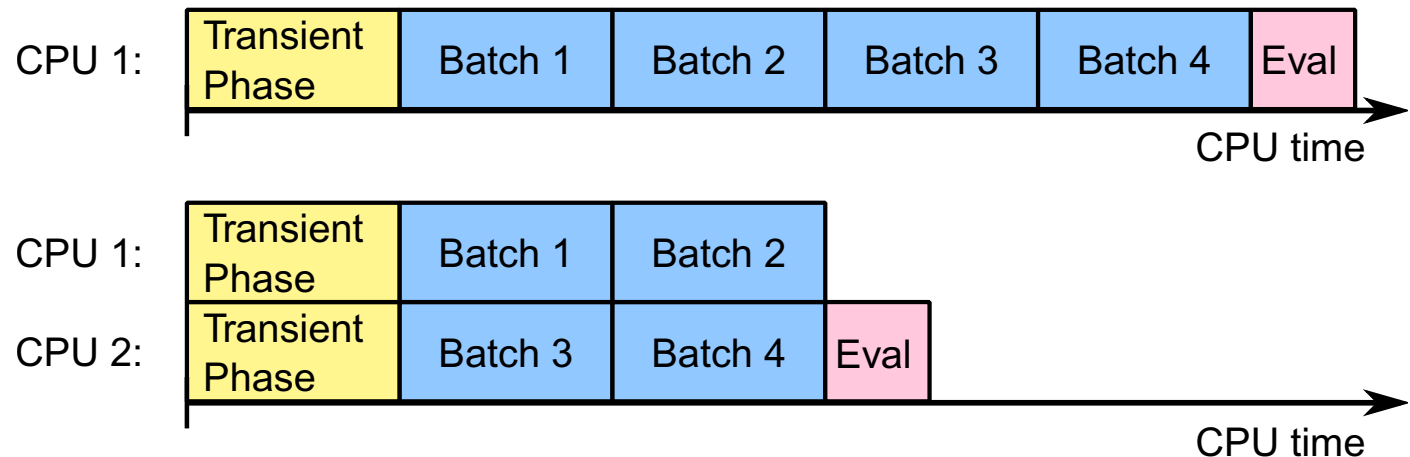
Advantages

- Easy to implement (existing code can be reused)
- Easy to debug

Problems

- Overhead due to transient phase
- Correct length of transient phase is more critical

- Instruction level
- Fine grained
- Task parallelism
- **Batch parallelism**
- Job offloading



→ Usable for simulation of mobile networks, implemented in IKR-Simlib

Parallelism

Job offloading (1)

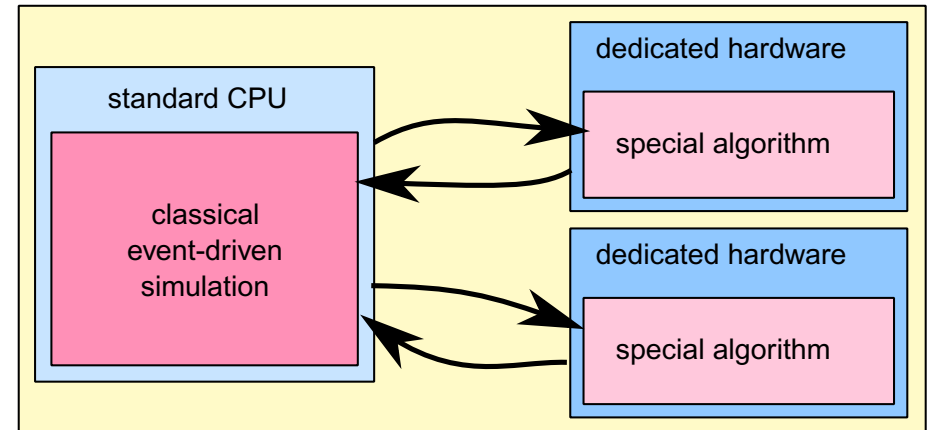
Basic idea

- Run main simulation on normal computer
- Offload special jobs on dedicated hardware

Properties

- Only applicable for jobs that are long enough to hide communication overhead
- Taking advantage of dedicated hardware without changing the event-driven simulation paradigm

- Instruction level
- Fine grained
- Task parallelism
- Batch parallelism
- **Job offloading**



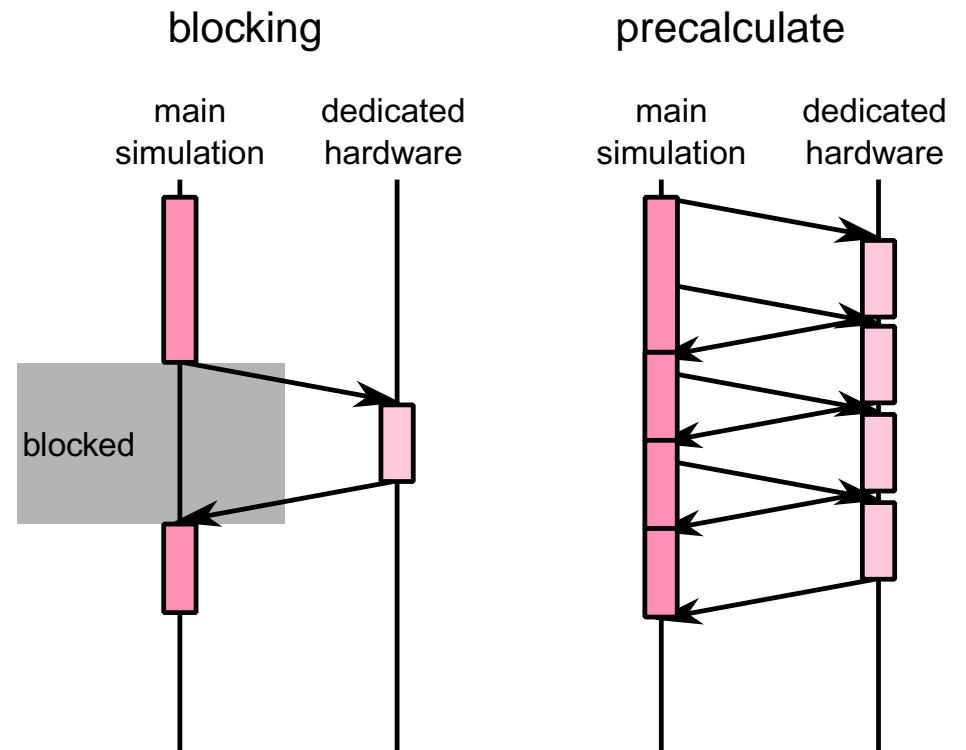
Parallelism

Job offloading (2)

Problems

- Offloading causes blocking that needs to be hidden
 - Hide blocking by running multiple instances in parallel
 - Precalculate where possible
- Debugging is harder

- Instruction level
- Fine grained
- Task parallelism
- Batch parallelism
- **Job offloading**



→ Suitable for certain problems (radio channel calculation, optimization ...)

Outline

- Motivation
- Current situation
- Parallelism
 - Overview
 - Types of parallelism
 - Instruction level
 - Fine grained
 - Task parallelism
 - Batch parallelism
 - Job offloading
- Hardware architectures
 - IBM Cell processor
 - NVIDIA Cuda
- Conclusion

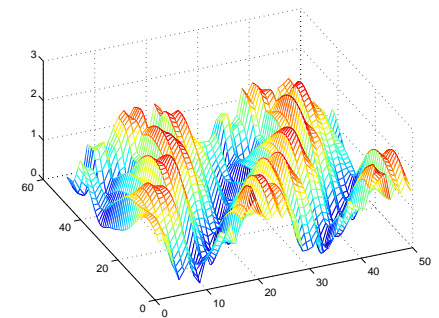
Radio Channel Calculation

Computational complex algorithm as example

- Channel model provides the "taps" by specifying θ_n , f_{D_n} , and τ_n
- Depending on channel model, these are constants or computational complex random processes
- Fourier-Transform:

$$H(\omega) = \sum_{n=1}^N e^{j\theta_n} e^{j2\pi f_{D_n} kT} e^{-j\omega\tau_n}$$

- Computed for each combination of mobile and base station, each TTI, multiple frequencies
- Model implemented on dedicated hardware in the following examples:
WSSUS (Hoeher model)



NVIDIA CUDA

Overview

Properties

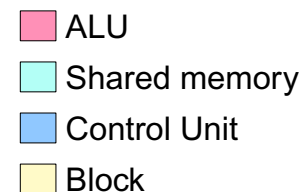
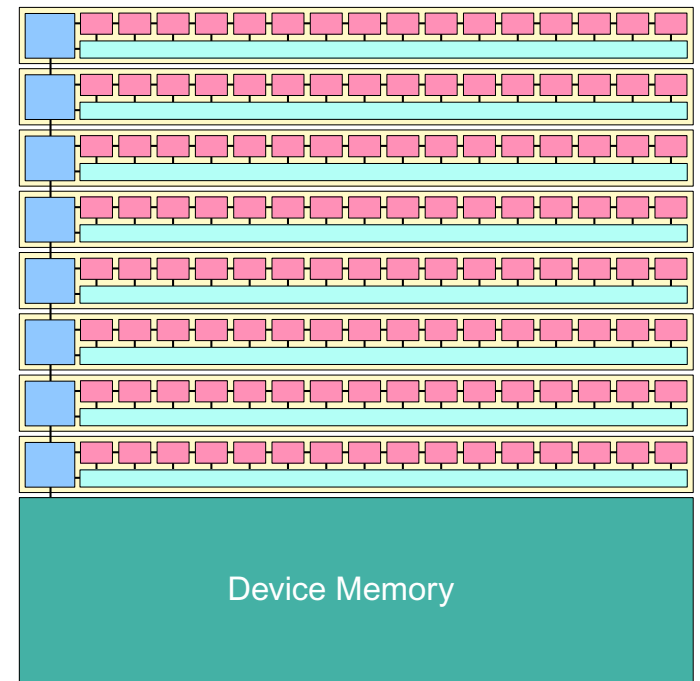
- 96 processing units
- Organizes in blocks
- Memory access and command execution inside blocks needs to be synchronized
- Local memory very limited
- Processing units support fast and transparent multithreading
- Transfer from/to device memory has to be hidden with threads

Constraints on suitable problems

- Multiple (>1000) problems that need the same sequence of commands to be solved
- Or single big problem that can be divided into multiple such problems



Above image from http://www.nvidia.com/object/geforce_8800gt.html



NVIDIA CUDA

Radio channel computation

Main Aspects

- Using CUDA (API to program NVIDIA graphics cards in C)
- Synchronous kernel, precalculating channel data

Main Problems

- Memory limitations on GPU
- No asynchronous communication

Performance

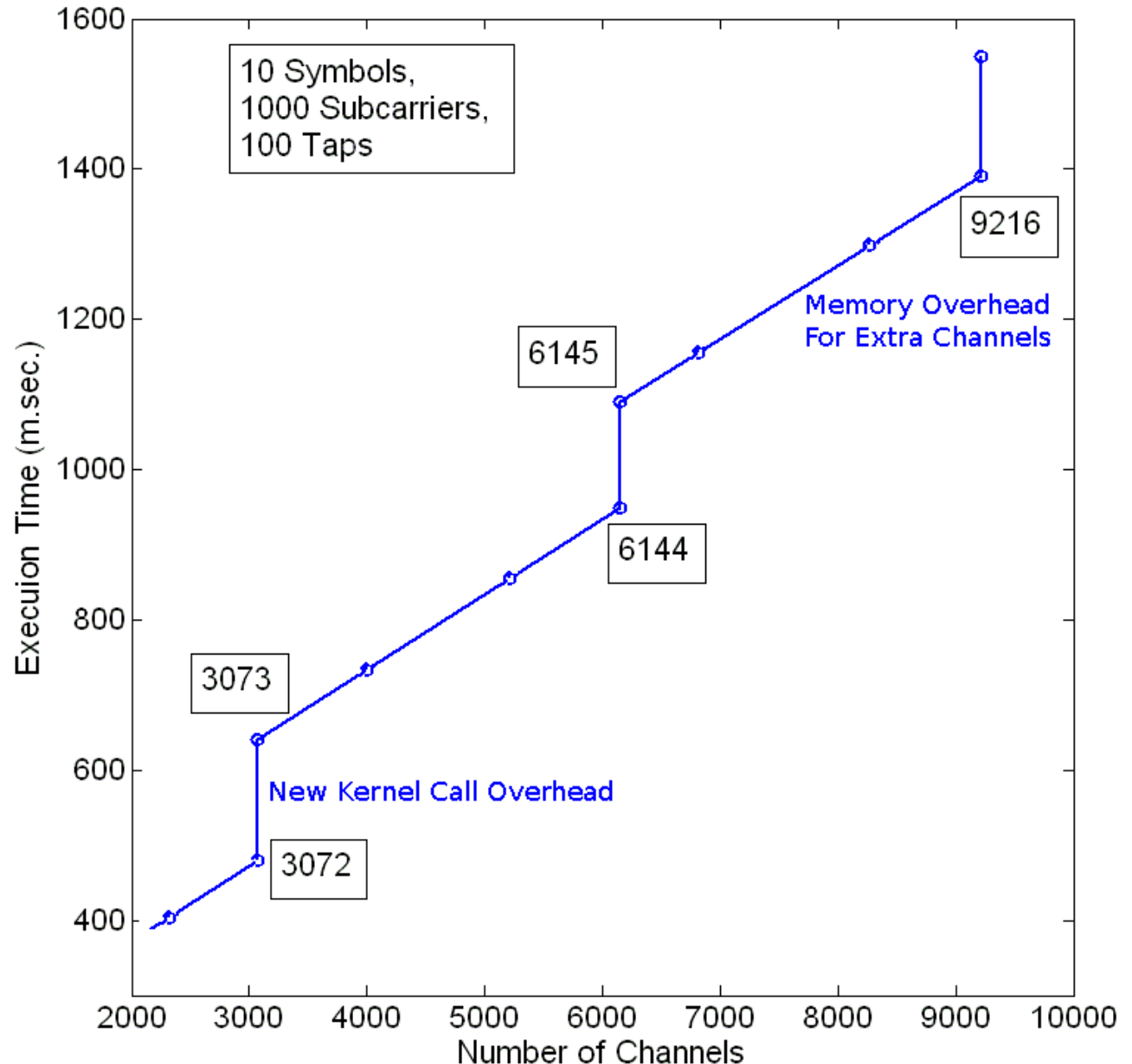
- ~30x speedup over 2.4 GHz Intel Core 2
- Roundtrip time for 1 kernel call around 1 second

NVIDIA CUDA

Results

Graph shows:

- X axis: number of channels to be calculated in the simulation
 - Y axis: total execution time (normalized)
 - Main simulation is not considered in this graph (dummy simulation)
- Each kernel call has to solve thousands of jobs at the same time (threading)



IBM Cell Processor

Overview

Properties

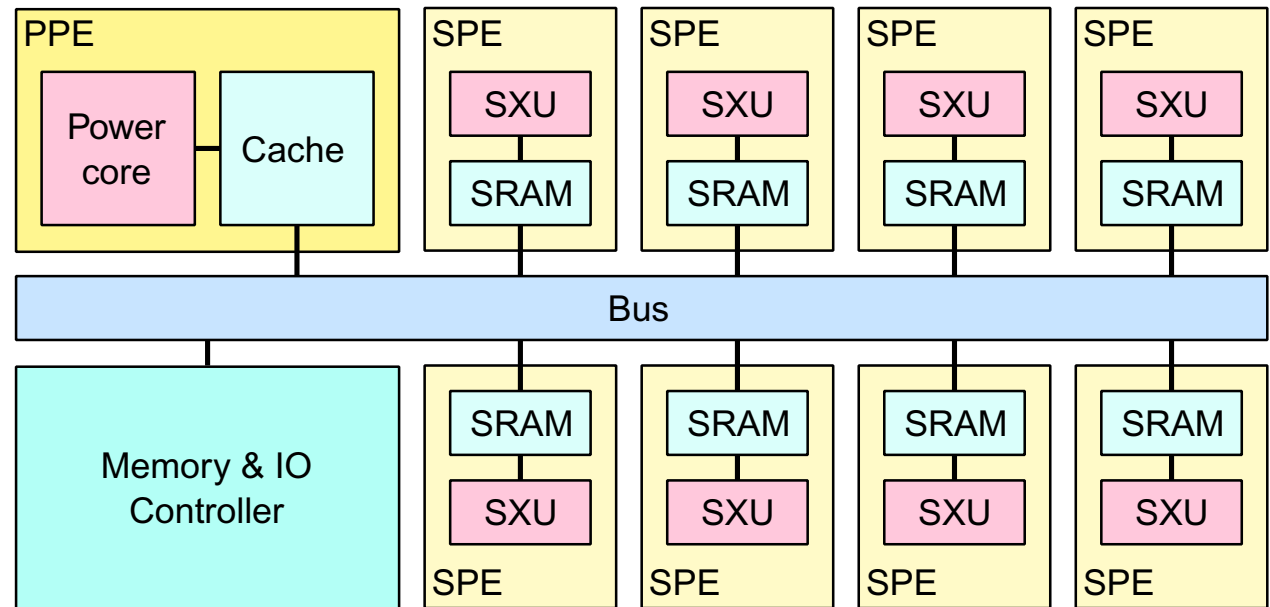
- ~200 GFlops
- Each SPE contains
 - 256 kB SRAM
 - 4x SIMD (single precision)
 - No MMU etc.
- PPE & SPEs communicate over DMA
- 8 or 6 SPEs per chip



Above image from <http://www-06.ibm.com/systems/jp/bladecenter>



Above image from <http://www.ps3informer.com/playstation-3/news/>



DMA Direct Memory Access
SPE Synergistic Processing Element
MMU Memory Management Unit (Address calculation etc.)

PPE PowerPC Processing Element
SXU Synergistic eXecution Uni

IBM Cell Processor

Radio channel computation

Main Aspects

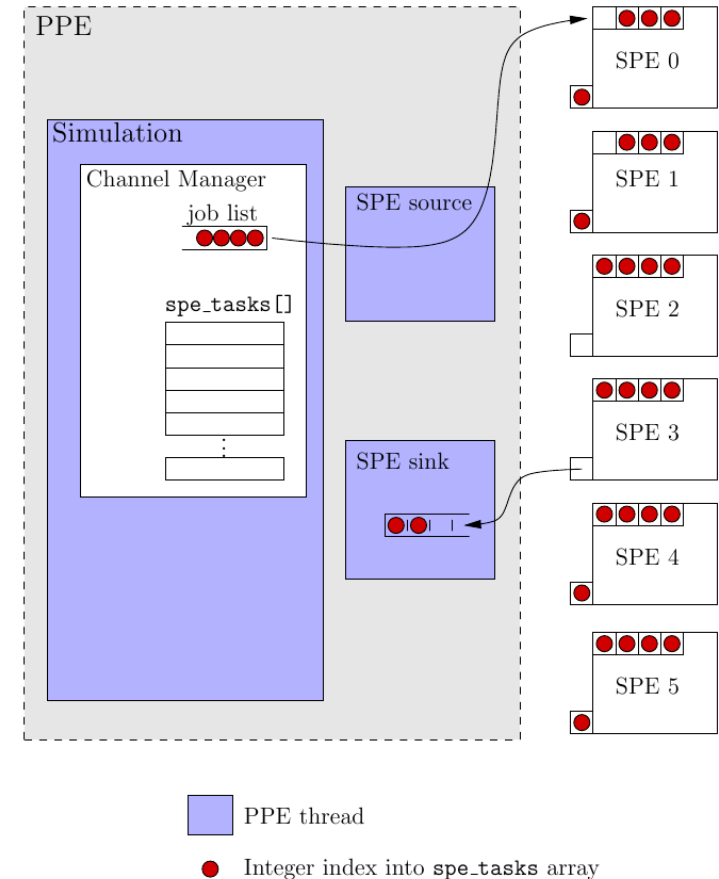
- Kernel on each SPE
- Asynchronous communication with PPE
- Hand-Optimizing code:
 - Vectorization for SIMD instruction set done by hand
 - SXU pipeline by hand

Main Problems

- Parallel programming (mutex debugging)
- Scheduling of jobs to PPEs

Performance

- ~16x speedup over 2.2 GHz AMD Opteron
- Roundtrip time for PPE-SPE communication ~3 μ s (under full load)

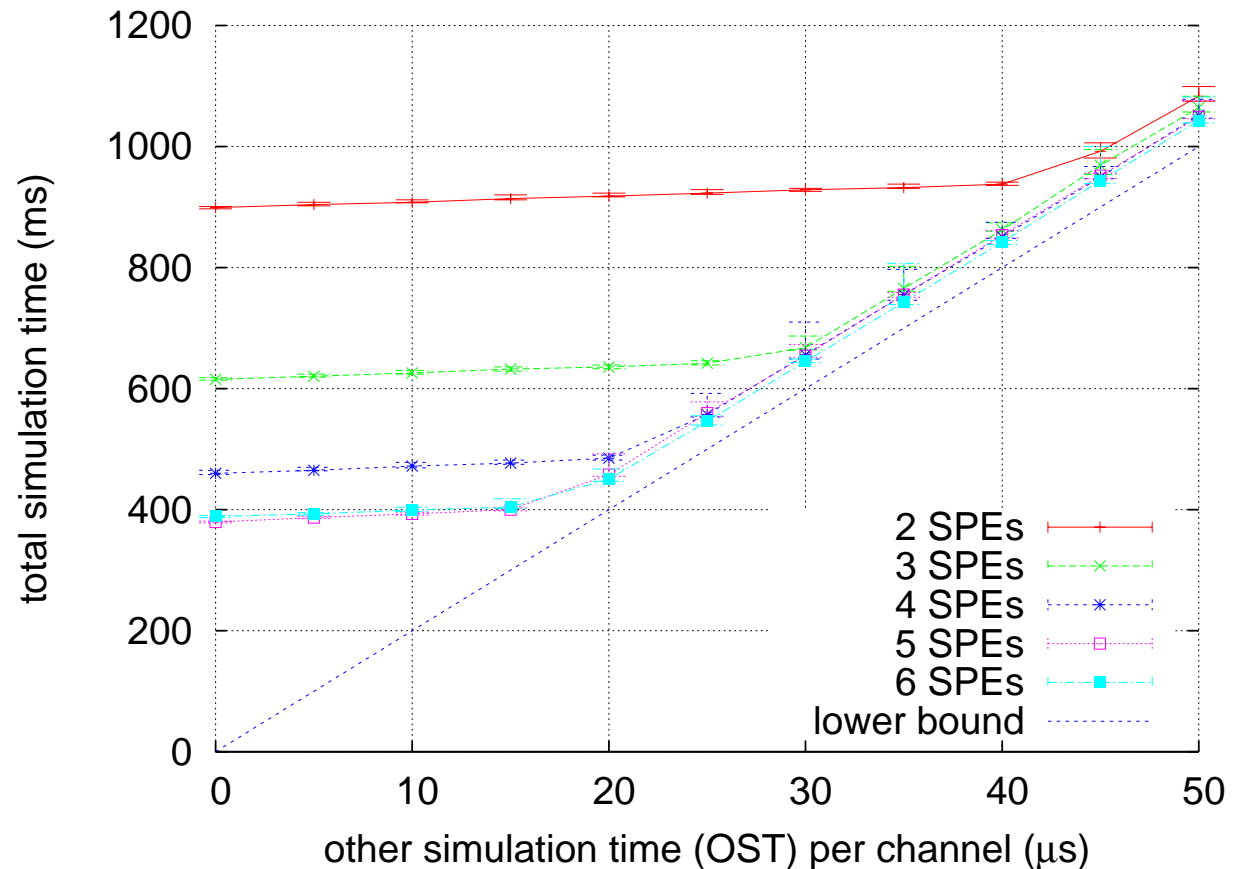


IBM Cell Processor

Results

Graph shows:

- X axis: real time, the main simulation needs per TTI and channel
- Y axis: total execution time (normalized)



→ Optimal performance when the computation time in the main simulation and on SPEs are similar (both run 100% of time)

Job Offloading


Architecture comparison

| | x86 Computer | Cell Processor | NVIDIA CUDA |
|----------------------------|----------------------------------|--------------------------------------|------------------------|
| Performance | ~4 GFlops per core | ~200 GFlops | ~500 GFlops (8800 GTX) |
| Typical latency | ~1us | ~3 us | typically Seconds |
| Memory per Thread | GBytes | 200 kByte | 0.5 .. 10 kByte |
| Programming | All languages | Full C/C++ with intrinsics or ASM | Basic C with addons |
| Paralleization methods: | SIMD, Batches, Job offloading | Job offloading | Job offloading |
| Price (06/2008) | ~300 EUR per core | PS3: ~450 EUR Blades: ~5000 EUR | ~100 EUR |

Conclusion

Problems

- Message passing
 - Classical event driven simulation
- Physical models
 - Channel models
 - User behavior
 - Raytracing
- Algorithms
 - Optimization: LP, ILP, GA, SA, ...
 - Graph algorithms: routing, coloring
 - Machine learning: classification, pattern recognition
 - Signal processing



Batch parallelization
New simulation paradigms
SIMD
Job Offloading
Cell Processor
NVIDIA CUDA

Conclusion

Parallelization methods

SIMD

Limited performance (max. 2 .. 4) gain, vectorization required

Fine Grained Parallelism

New paradigm required, complete rewrite of existing code

Task parallelism:

Only suitable for certain kinds of simulation, synchronization problem

Job Offloading

Suitable for certain problems as an add-on to unmodified simulation

Batch parallelism

Suitable, implemented in IKR-Simlib