Convey Vector Personalities – FPGA Acceleration with an OpenMP-like programming effort?

Björn Meyer, Jörn Schumacher, Christian Plessl, Jens Förstner
University of Paderborn, Germany

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

PC² PADERBORN CENTER FOR PARALLEL COMPUTING

FPGA-accelerated Computing in Science

- motivation
  - science increasingly depends numerical simulation
  - many scientists are open for new HPC technologies
- research has shown high potential of FPGA acceleration
  - high and predictable sustained performance
  - ultimate control over computation and communication
  - supports many kinds parallelism and customization
- challenges
  - FPGAs are way too hard to program – even for many computer scientists
  - synthesis, place and route, verification and debugging still is a world of pain

Enabling Scientists to use FPGA Accelerators

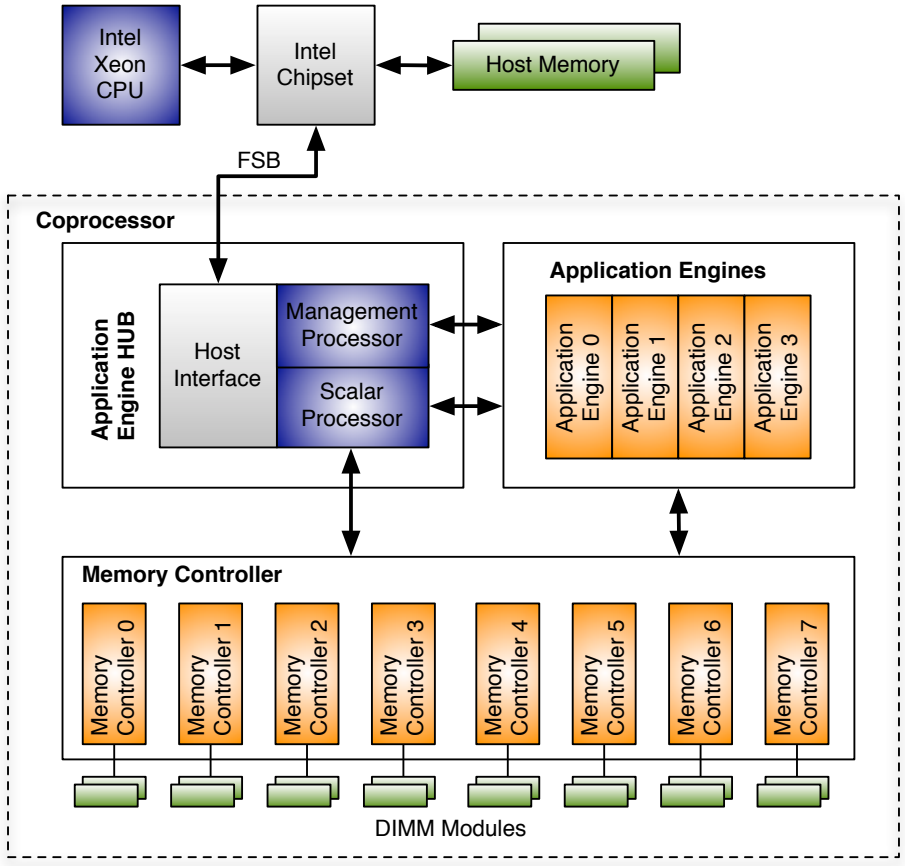
- current tool flows
  - HDL-based
    - not targeted a scientific users, way too cumbersome
    - missing integration of platform and tools
  - general high-level synthesis tools
    - frequently not sufficiently automated, incomplete support of language

- compiler directive approaches
  - developer annotates code sections to be accelerated with #pragmas
  - completely automated partitioning and tool flow
  - successfully used for multi-cores (OpenMP) and GPUs (HMPP,PGI,...)
  - recent standardization efforts (OpenACC)

- Convey offers a directive-based FPGA acceleration solution
  - goal of this paper: evaluate performance and usability
  - compare with parallel implementation on multi-cores

Convey HC-1 Hybrid Core Computer

- integrated FPGA accelerator system for HPC
  - 4 user-programmable application FPGAs (Virtex-5 LX330)
  - tightly coupled with host CPU
- memory subsystem
  - cache coherent shared memory between CPU and FPGAs
  - 80GB/s max. bandwidth for co-processor (8 independent memory banks)
  - no caches on co-processor
- firmware (personalities) for HPC application domains
  - bioinformatics
  - computational finance
  - vector processing
  - custom personality



Convey Vector Personality

- programmable vector processor implemented in FPGA
  - acceleration of scientific codes with SIMD parallelism
  - 64 vector registers, 1024 elements per register
  - up to 1024-wide vector operations
  - basic mask support (loads and stores only)
  - variant for single and double precision arithmetic

- vectorizing compiler
  - supports C, C++ and Fortran
  - code to be executed on vector processer can be marked with directives
    - host code translated to x86 instructions
    - accelerator code translated to vector instructions
    - transparent application partitioning, data transfers, synchronization
  - no custom hardware generation

Acceleration Directives

- acceleration directives very similar to OpenACC or OpenMP
- basic acceleration by wrapping code in `begin_coproc/ end_coproc #pragmas`
- additional pragmas for optimization
  - optimization of data transfer
  - guiding compiler optimization

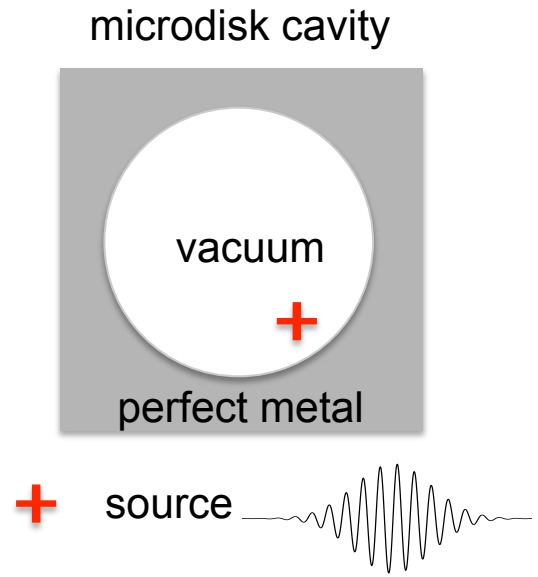
```
#pragma cny migrate_coproc(x, xByteSize);
#pragma cny migrate_coproc(y, yByteSize);
#pragma cny begin_coproc
unsigned long i ;
#pragma cny no_loop_dep ( y )
for (i = 0; i < size; i++) {
  y[i]=a*x[i]+y[i];
}
#pragma cny end_coproc
```

example: SAXPY

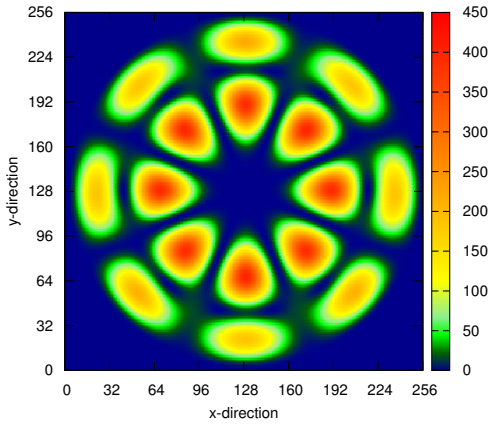
Case Study: Computational Nanophotonics

- computational nanophotonics
  - collaboration with theoretical physicists
  - study light field in nanostructured materials
  - computationally expensive (hours to weeks per simulation)

- microdisk cavity in perfect metallic environment
  - well studied nanophotonic device
  - point-like time-dependent source (optical dipole)
  - known analytic solution (whispering gallery modes)



result: energy density

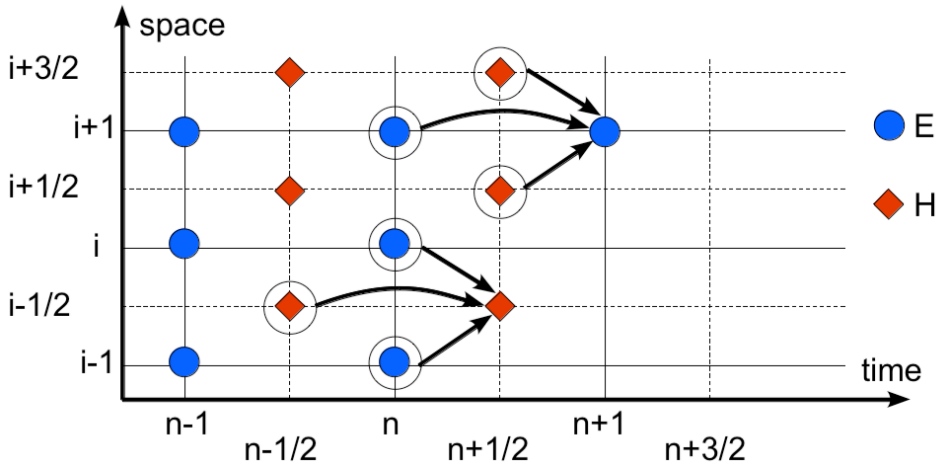


Finite Difference Time Domain Method (FDTD)

- numerical method for solving Maxwell's equations
- iterative algorithm, computes propagation of fields for fixed time step
- stencil computation on regular grid
  - same operations for each grid point
  - fixed local data access pattern
  - simple arithmetic
- difficult to achieve high performance
  - hardly any data reuse
  - few operations per data

$$
\begin{aligned}
E_x[i] &= ca \cdot E_x[i] + cb \cdot (H_z[i] - H_z[i - dy]) && (1)\\
E_y[i] &= ca \cdot E_y[i] + cb \cdot (H_z[i - dx] - H_z[i]) && (2)\\
H_z[i] &= da \cdot H_z[i] + db \cdot && (3)\\
&\quad \cdot (E_x[i + dy] - E_x[i] + E_y[i] - E_y[i + dx])
\end{aligned}
$$

FDTD update equations
(for one time step)



Evaluation of Vector Personalities and Toolflow

- compare 3 implementation of computational nanophotonics case study
  1. naive OpenMP-parallelized implementation for multi-core
  2. optimized OpenMP-parallelized implementation for multi-core
  3. Convey HC-1 vector personality
- metric: grid-point updates/s
- hardware platforms

	Convey HC-1	Convey HC-1	Workstation
CPU	Xeon 5138	Xeon L5408	Xeon E5620
Clock	2.13 GHz	2.13 GHz	2.4 GHz
Cores	2	4	2 x 4
Cache	4 MB	6 MB	2 x 12 MB
Memory	24 GB	64 GB	12 GB
DIMM type	standard	scatter-gather	standard
Price	~50'000$	~50'000$	~5'000$

Naive OpenMP implementation

- direct implementation of FDTD iteration equations
- basic cache optimizations
  - NUMA aware memory allocation (first touch policy)
  - always process same part of data with same core by using static OpenMP scheduling
  - avoid cache line invalidation by using different arrays for reading and writing (similar to double buffering)

    E' [i] = f(E[i],H[i],H[i-dx]) instead of E[i]=f(E[i],H[i],H[i-dx])

OpenMP Implementation w/ Spatial Tiling (1)

- in addition to optimization from naive implementation...
- ... use spatial tiling (cache blocking) to improve multi-core cache performance
- tile x- and y-dimensions are open parameters

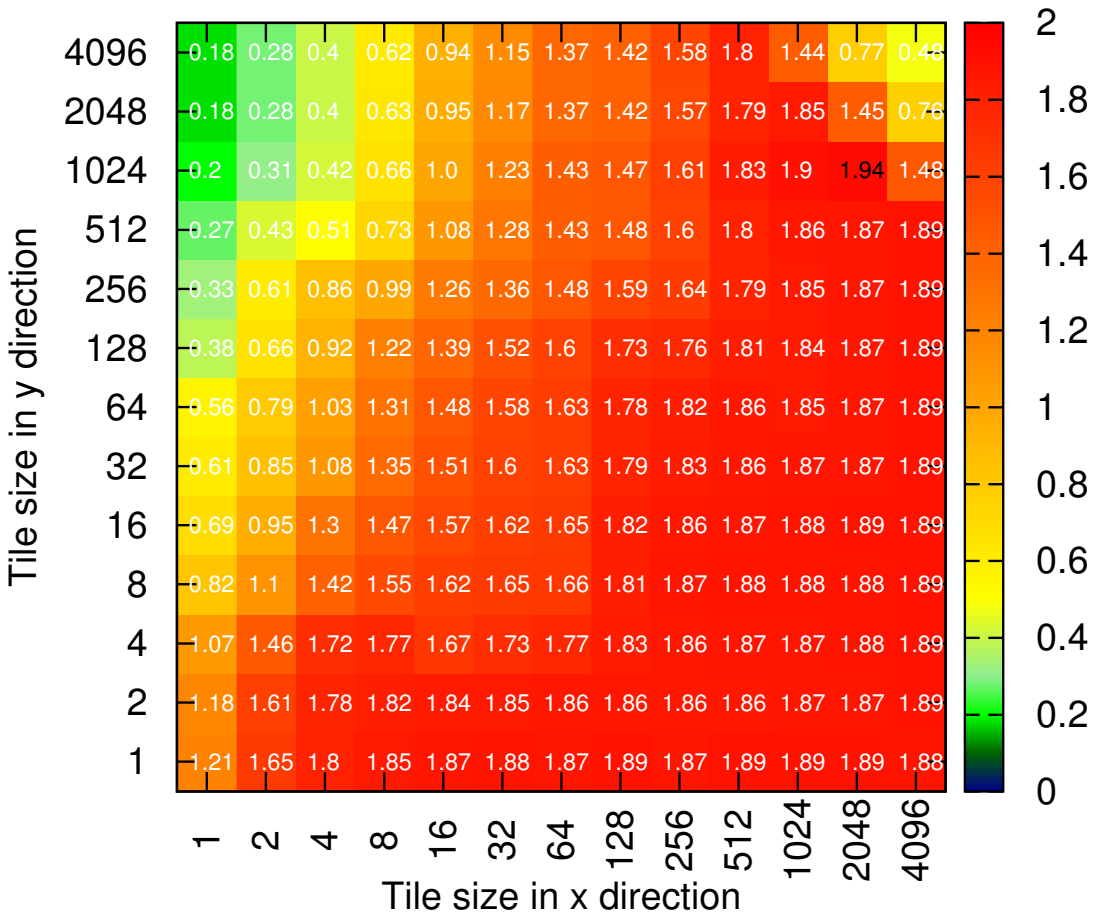
```
#pragma omp parallel for schedule(static) private(minusX, minusY, i)
collapse(2)

for (int yy = 1; yy < yDim-1; yy+=ty) {
  for (int xx = 1; xx < xDim - 1; xx+=tx) {

    for (int y = yy; y < MIN(yy+ty,yDim - 1); y++) {
      for (int x = xx; x < MIN(xx+tx,xDim - 1); x++) {
        i = x + y*xDim;
        minusX = i-1;
        minusY = i-xDim;
        ExNext[i] = ca * Ex[i] + cb * (Hz[i] - Hz[minusY]);
        EyNext[i] = ca * Ey[i] + cb * (Hz[minusX] - Hz[i]);
      }
  }
}}
```

OpenMP Implementation w/ Spatial Tiling (2)

- determine optimal tile size with auto-tuning



speedup over naive implementation (2D double precision, 4096x4096 grid)

Convey HC-1 Vector Personality Implementation

- based on naive implementation (no tiling since no caches)
- move loops that update electromagnetic fields to accelerator
- optimizations
  1. NUMA-aware data structure placement: co-processor can access its local memory faster than the CPU memory
  2. loop unrolling: unroll loops 16 times
  3. remove needless memory barriers: co-processor uses a weakly ordered memory model, compiler is too pessimistic, inserts superfluous barriers
  4. rewrite inner loop to enable vectorization (manual masking operation)

```
for(i=0..N){
  if (is_in_vacuum(i)){
    H[i] = ...
  } else {
    H[i] = 0
  }
}
```

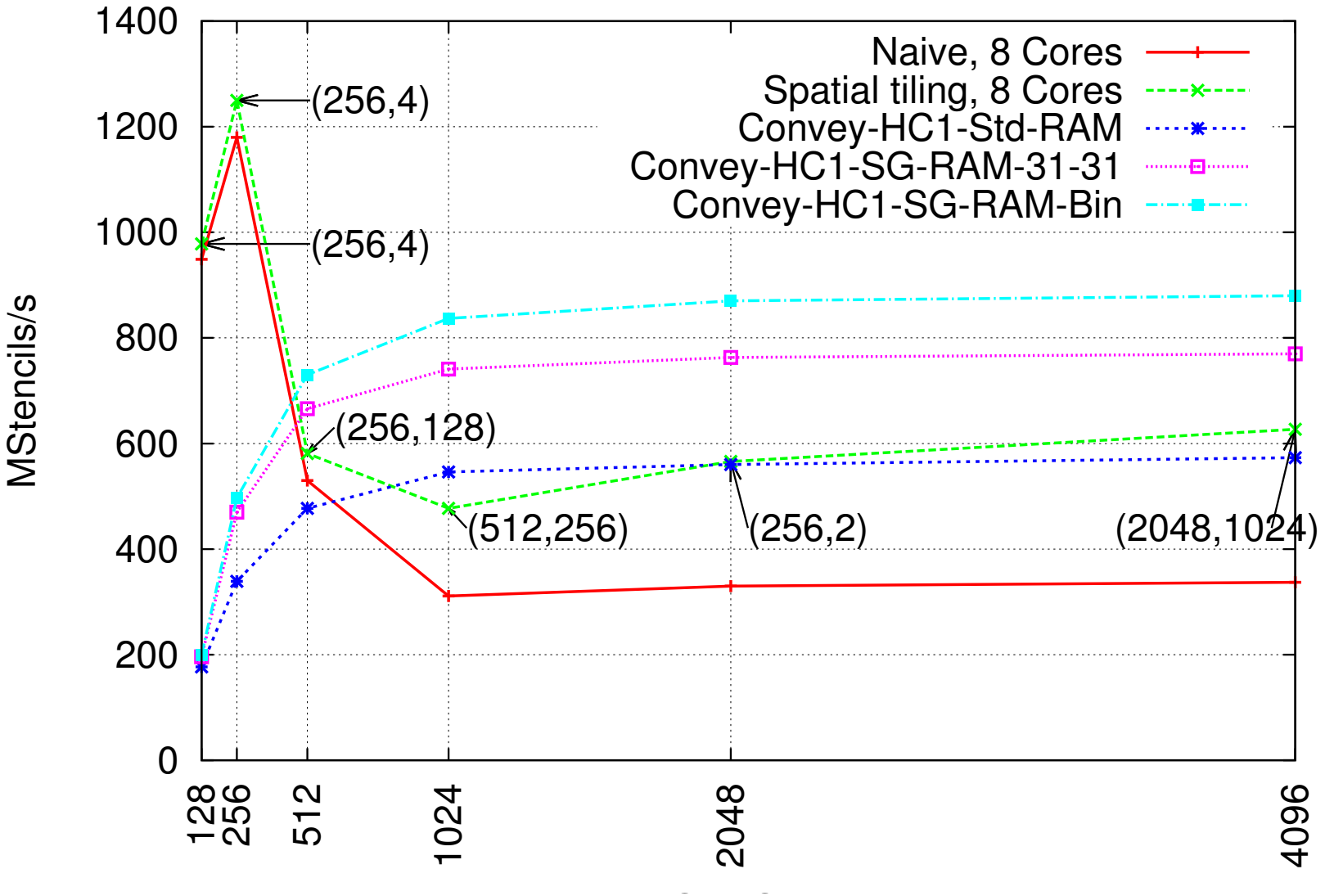
conditional prevents vectorization

```
// vaccum[i] = 1 if i not in vaccum
// vaccum[i] = 0 if i in vaccum

for(i=0..N){
  H[i] = ...
  H[i] = H[i] * vaccum[i]
}
```

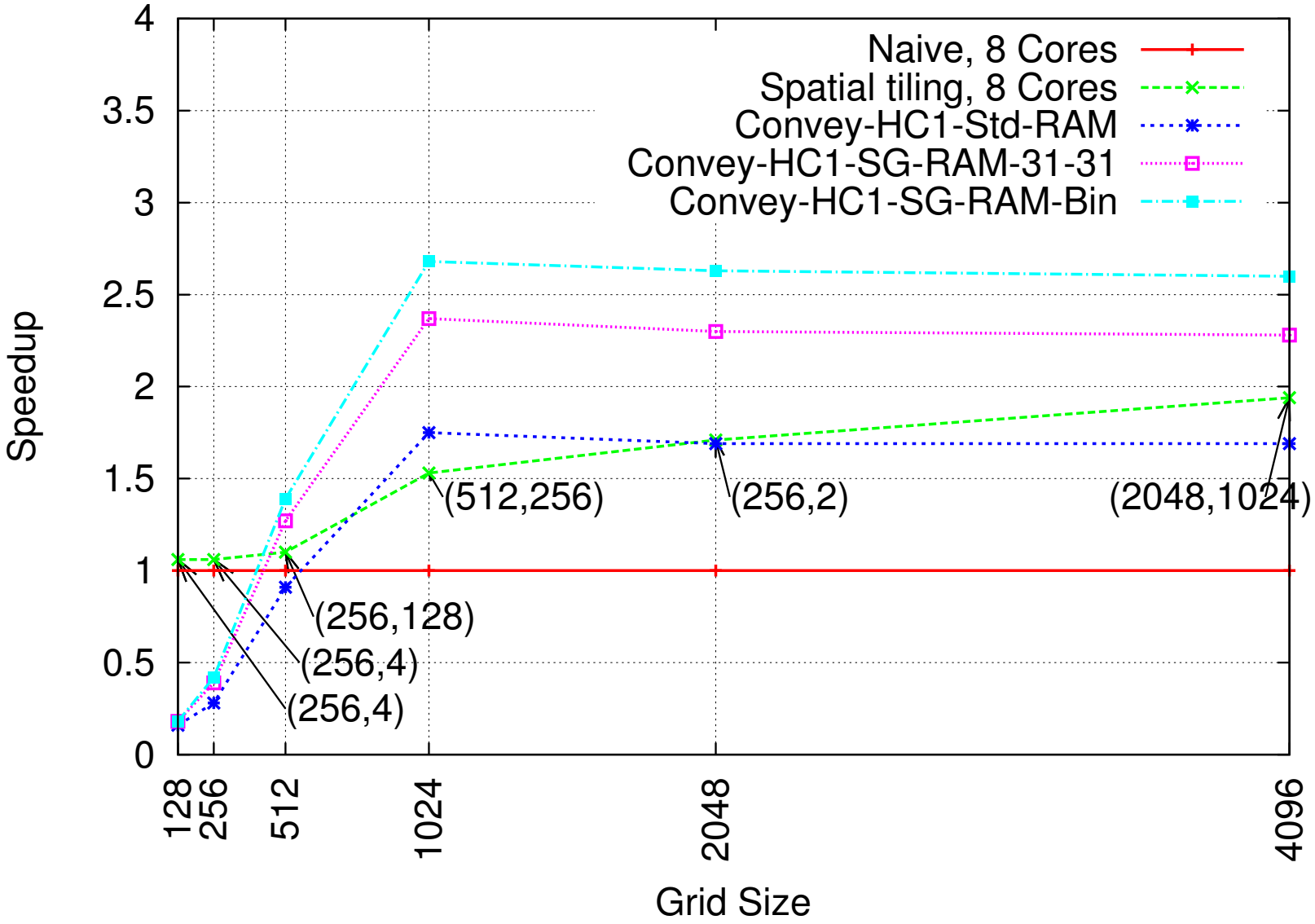
can be vectorized

2D Absolute Performance

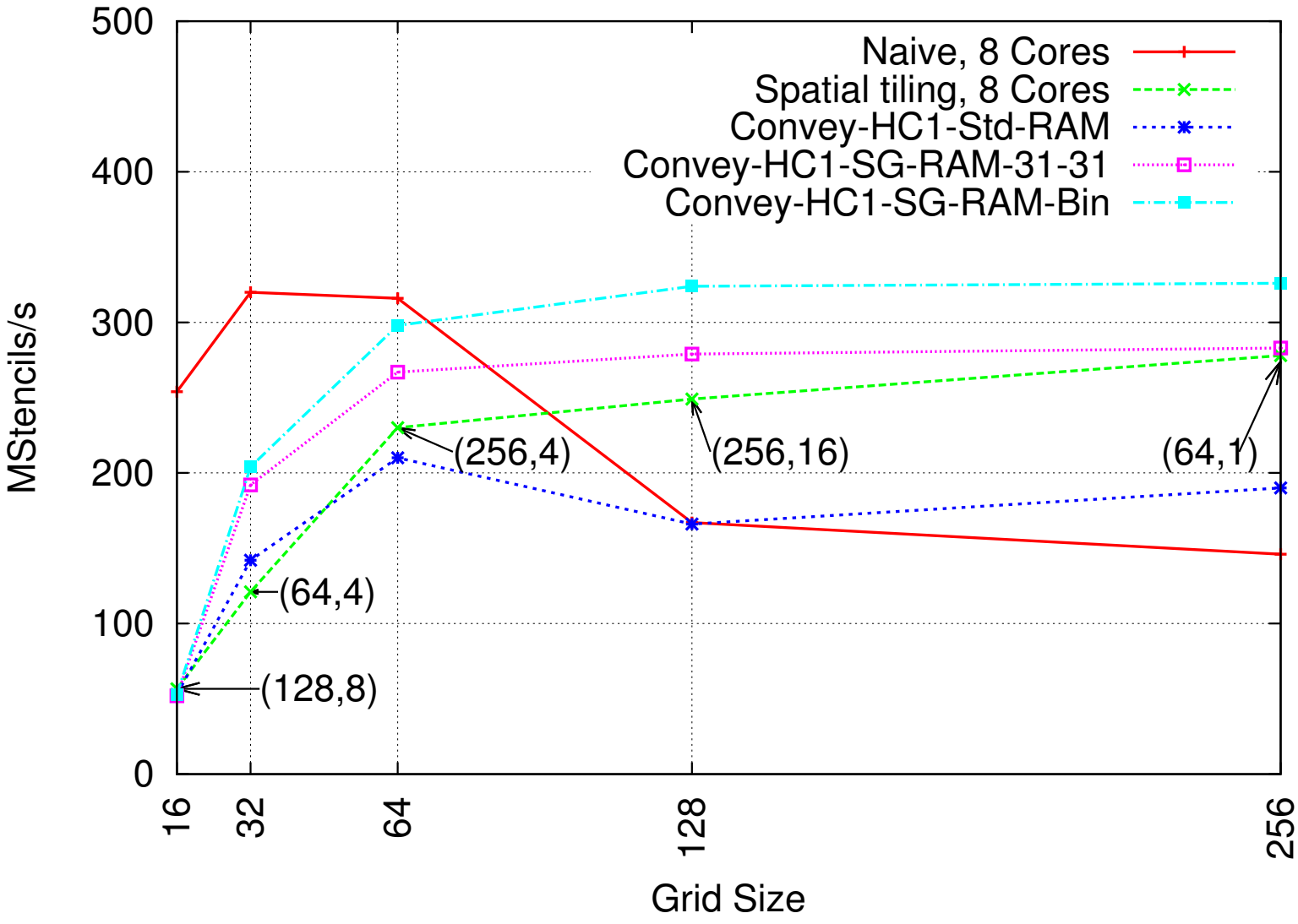


working set size for grid size 512 * 512:
(512*512) * 3 fields * 2 (double buffering) * 4 (double precision) = 6MB

2D Relative Performance / Speedup

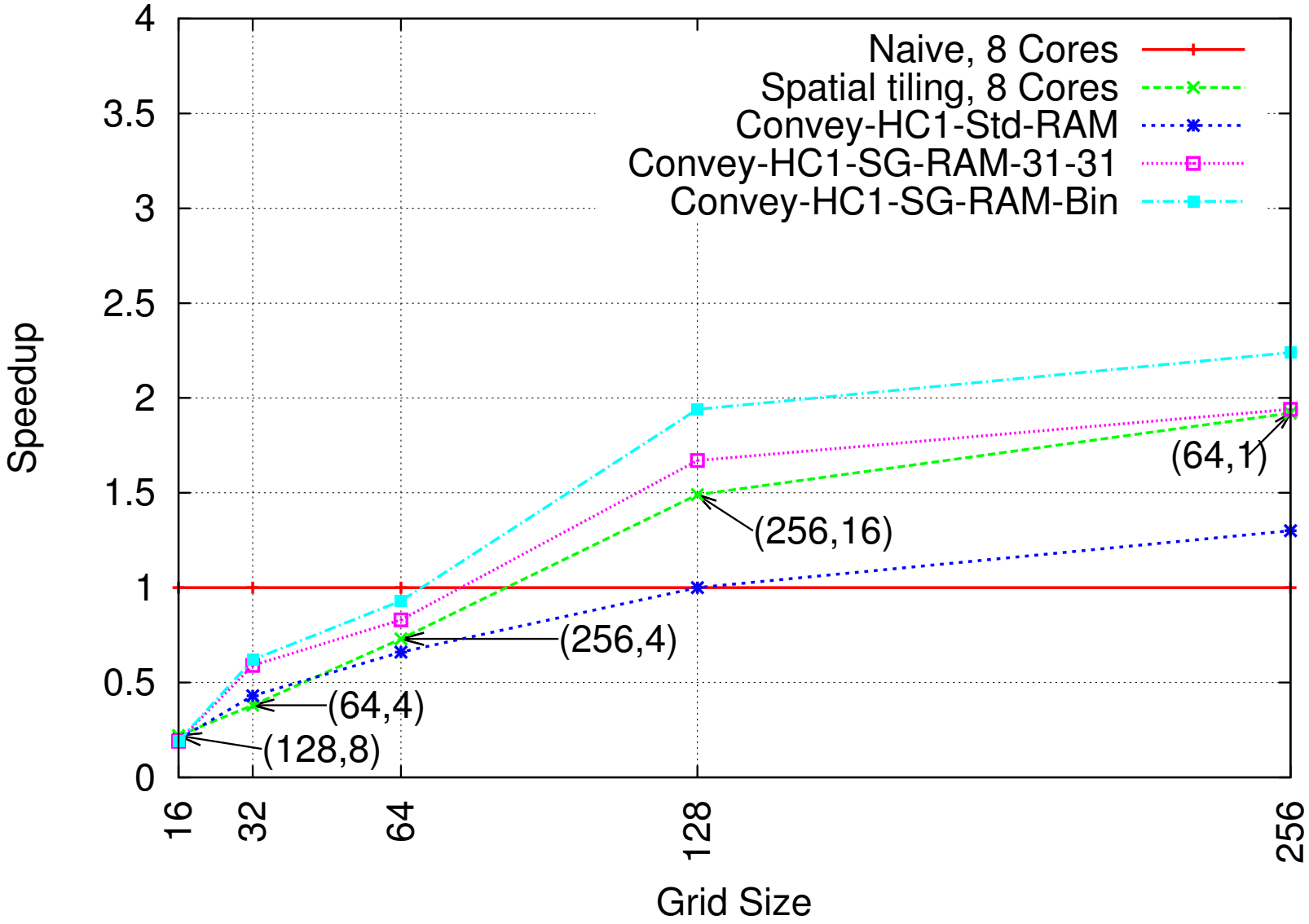


3D Absolute Performance



working set size for grid size 64*64*64:
(64*64*64) * 6 fields * 2 (double buffering) * 4 (double precision) = 12MB

3D Relative Performance / Speedup



Is it Worth it?

- productivity
  - working implementation in minutes
  - significant performance gain over single-core CPU
  - performance benefit decreases when comparing against optimized, parallel multi-core implementation

- obtaining good performance is difficult
  - crippled mask support in vector processor hampers performance
  - quality of the compiler (cnyCC) is unfortunately rather poor
    - poor optimizations
    - very dependent on specific form of code
  - meanwhile Convey has released a new compiler (external vectorizer), which is – at least in our setup – unusable

Is it Work it? (2)

- economics
  - for our case study the HC-1 vector personality was the fastest implementation
    - 2D: HC-1 outperforms SMP multi-core by 40%
    - 3D: HC-1 outperforms SMP multi-core by 17%
  - development effort comparable to OpenMP solution
  - difference in price of HC-1 and SMP system ~10x

Conclusions & Outlook

- directive-based acceleration has many benefits from a usability perspective
  - Convey has shown that such a tool flow is generally possible
  - stencil computations run about 20-40% faster than on multi-core SMPs, tough economically this use case is questionable
- implementing a generic processor may not be the best use of FPGA resources
  - doesn't exploit customization potential
  - low average utilization
- open question: can we generate optimized processors from application that are augmented with compiler directives
  - continuum between fixed function and completely application specific

Questions & Feedback

Christian Plessl
University of Paderborn
Paderborn Center for Parallel Computing
christian.plessl@uni-paderborn.de