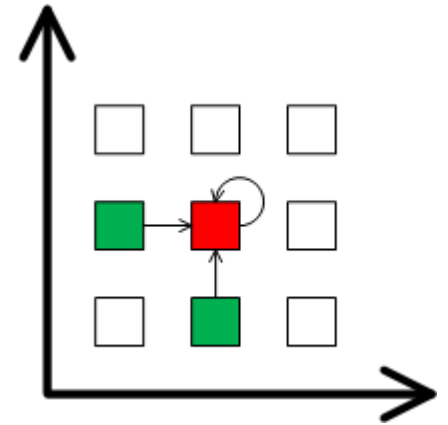


Solving the advection PDE on the Cell Broadband Engine

Georgios Rokos, Gerassimos
Peteinatos, Georgia Kouveli, Georgios
Goumas, Kornilios Kourtis and
Nectarios Koziris

Introduction

- Two-dimensional advection PDE
 - 3-point stencil operations
- Can be solved using
 - Gauss-Seidel-like solver (in-place algorithm)
 - Jacobi-like solver (out-of-place algorithm)
- Performance depends on:
 - Efficient usage of computational resources
 - Available memory bandwidth
 - Processor local storage capacity
- Platform of choice for experimentation:
 - Cell Broadband Engine



Cell Broadband Engine

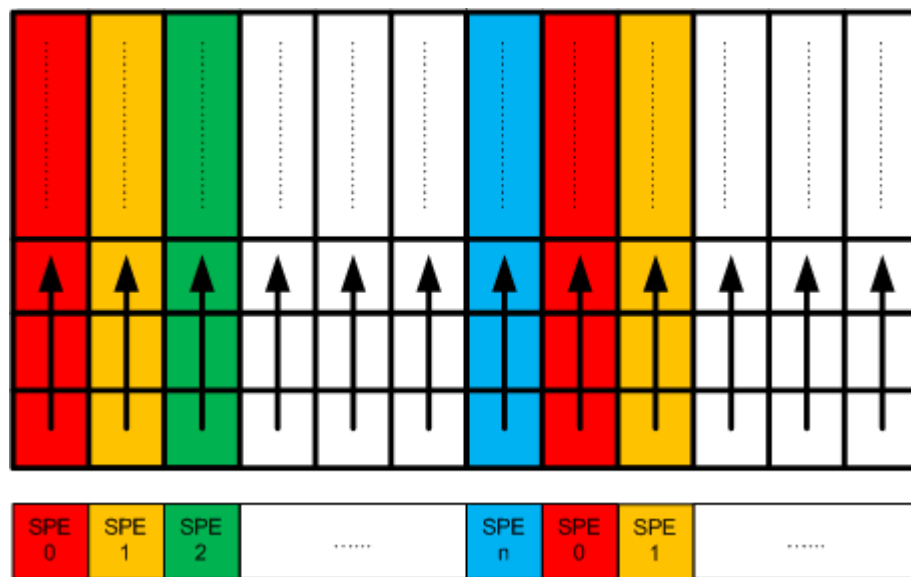
- Heterogeneous, 9-core processor
 - 1 PowerPC Processor Element (PPE) – a typical 64-bit PowerPC core
 - 8 Synergistic Processor Elements (SPEs) – SIMD processor architecture oriented towards high performance floating-point arithmetic
- Software-controlled memory hierarchy
 - No hardware controlled cache
 - Instead, each SPE has a 256 KB programmer-controlled local store
- Memory Flow Controller (MFC) on every SPE
 - Supports asynchronous DMA transfers
 - Can handle many outstanding transactions
- Processing elements communicate via high-bandwidth Element Interconnect Bus (EIB)
 - 20.46 GB/s
 - Provides the potential of more efficient usage of memory bandwidth

Motivation

- Evaluate Cell B/E as a platform for executing the advection PDE solver
- Explore optimization techniques and determine the contribution of each one to execution performance
- Compare in-place and out-of-place versions of the solver in terms of:
 - raw performance
 - total completion time (convergence rate / raw performance)
 - programmability

Implementation

- Blocking
 - Split matrix into blocks so that each one fits in the local store
 - Block boundaries have to be exchanged between neighboring processors
- Assignment of blocks to SPEs
 - Assign each SPE whole block-columns
 - This way, boundaries in the vertical direction are kept inside the SPE
 - Need to exchange boundary values only in the horizontal direction



Optimizations

- Multi-buffering
 - Transfer old / new blocks to / from memory while performing computations on current block, overlap computation / communication
 - CBE provides the option of using asynchronous DMA transfers
- Vectorization
 - Apply same operation to more than one data at once
 - SPE vector registers are 128-bit wide \Rightarrow 4 single-precision floating-point values in each vector
 - Theoretically, performance x4 for single-precision
 - In practice, benefits are higher than that since SPEs are exclusively SIMD processors \Rightarrow manipulating scalar operands includes significant overhead
- Block-major layout
 - All block elements in consecutive memory addresses
 - Instead of standard C row-major order
 - Possible to transfer the whole block at once instead of row-by-row

Optimizations

- Instruction scheduling
 - Exploit heterogeneous pipelines to continuously stream data into the FP pipeline (even pipeline)
 - Load data in time using odd pipeline so that even pipeline does not stall waiting for them
 - Compiler tries to automatically accomplish this task; however, programmer has to assist the compiler by manually optimizing many parts of the application
- Block tiling
 - Group iterations into “super-iterations”
 - Exchange boundary values at the end of every super-iteration
 - More data are exchanged per transfer, since SPE has to send / receive boundary values for every iteration in the super-iteration group
 - But fewer transfers take place ⇒ less total communication overhead

In-place vs. Out-of-place

- Out-of-place algorithm
 - Jacobi-like approach
 - Uses neighbor values from **last** iteration
 - Known to be slower at convergence speed, since computation does not use the most up-to-date data
 - Data independence: easy to vectorize the algorithm

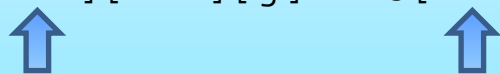
```
while(!converged())
{
  n = (++loops)%2;
  for(i = 1; i < Y; i++)
    for(j = 1; j < X; j++)
      U[1-n][i][j] = (1 + 2*a*dt/dx) * U[n][i][j] -
                    a*dt/dx * (U[n][i-1][j] + U[n][i][j-1]);
}
```



In-place vs. Out-of-place

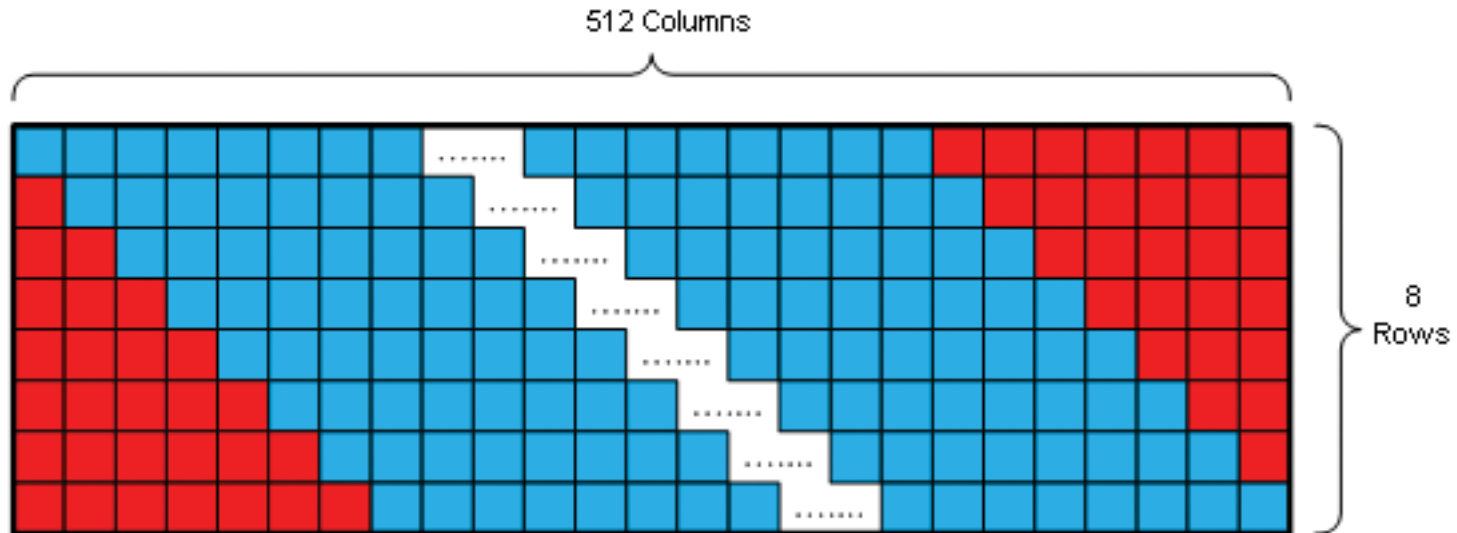
- In-place algorithm
 - Gauss-Seidel-like approach
 - Uses neighbor values from **current** iteration
 - Known to be faster at convergence speed, since computation uses the most up-to-date data
 - Data dependencies make vectorization difficult

```
while(!converged())
{
  n = (++loops)%2;
  for(i = 1; i < Y; i++)
    for(j = 1; j < X; j++)
      U[1-n][i][j] = (1 + 2*a*dt/dx) * U[n][i][j] -
                    a*dt/dx * (U[1-n][i-1][j] + U[1-n][i][j-1]);
}
```



In-place: Vectorization

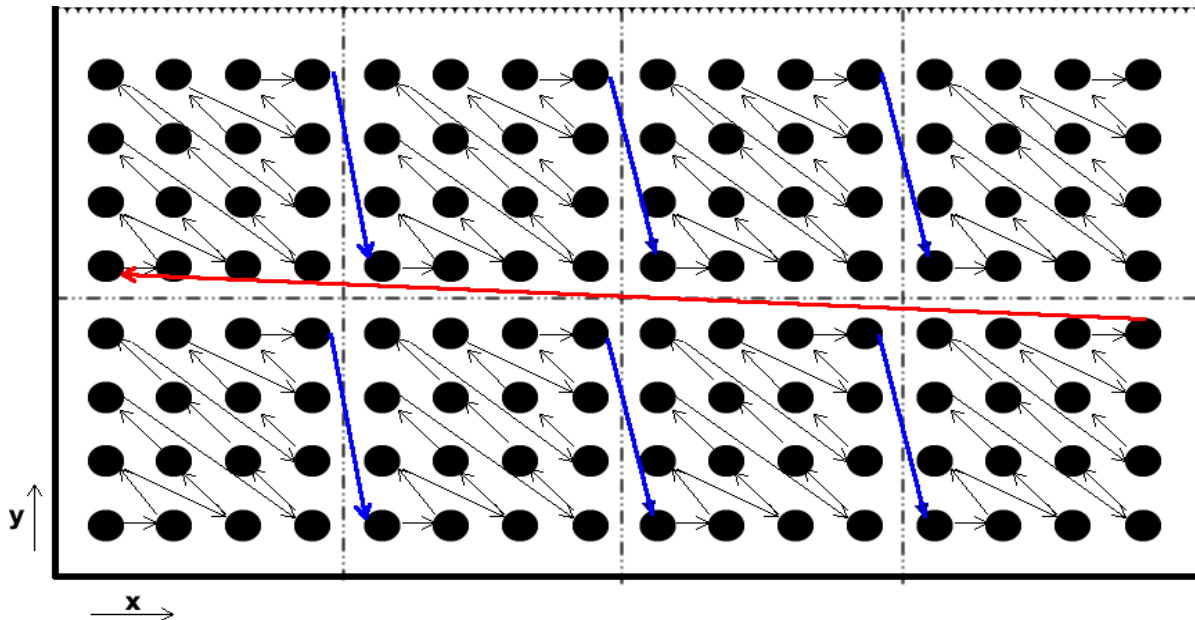
- Idea: traversing blocks in diagonal order
 - No dependence between elements in successive diagonals



- Diagonal traversal of block creates lead-in and lead-out areas
 - Difficult to vectorize \Rightarrow poor performance
 - Need to minimize them \Rightarrow elongated block shape
 - Experimentation: 8 x 512 was the best choice

In-place: Vectorization

- Problem: Diagonal elements not in consecutive memory addresses, need shuffling operations to form vectors
- Avoid shuffling each time the block is traversed
 - Permanently reorder elements in memory
 - Diagonal-major layout applied to each block separately

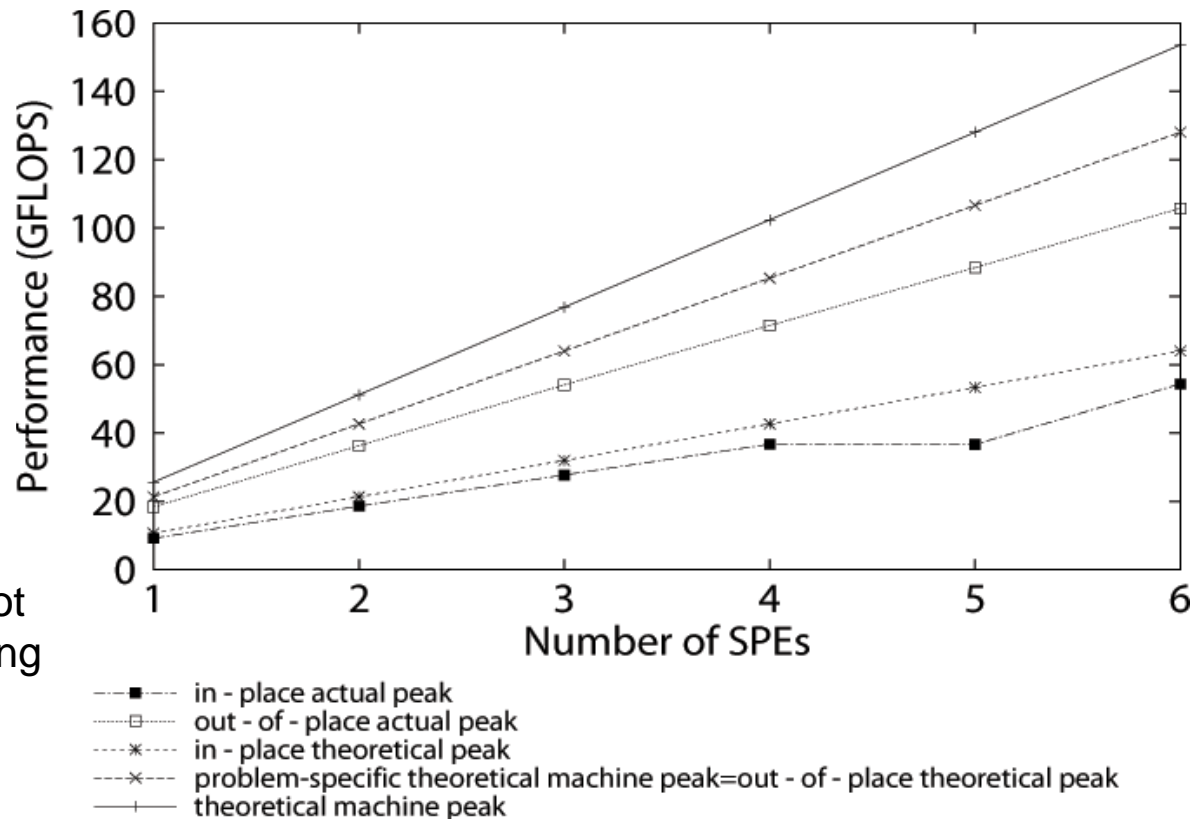


Experimental Evaluation

- Performed on a PlayStation3 console
 - 3.2 GHz Cell
 - 6 SPEs
 - 256 MB XDR RAM
 - Debian/GNU Linux – kernel 2.6.24
 - Cell SDK 3.1
- Measurements include
 - Performance in GFLOPS = $f(\# \text{ of SPEs})$
 - Total execution time = $f(\# \text{ of SPEs})$
 - Performance breakdown – contribution of each optimization technique

GFLOPS – Number of SPEs

- Out-of-place algorithm: performance results near theoretical peak
- In-place algorithm: performance results nearly half the theoretical peak
 - Data dependencies do not allow continuous streaming of data into the even pipeline
- Almost linear speedup for both algorithms
 - Good overlap of computation and communication
 - Divergence for 5 SPEs in in-place: due to uneven assignment of blocks to SPEs

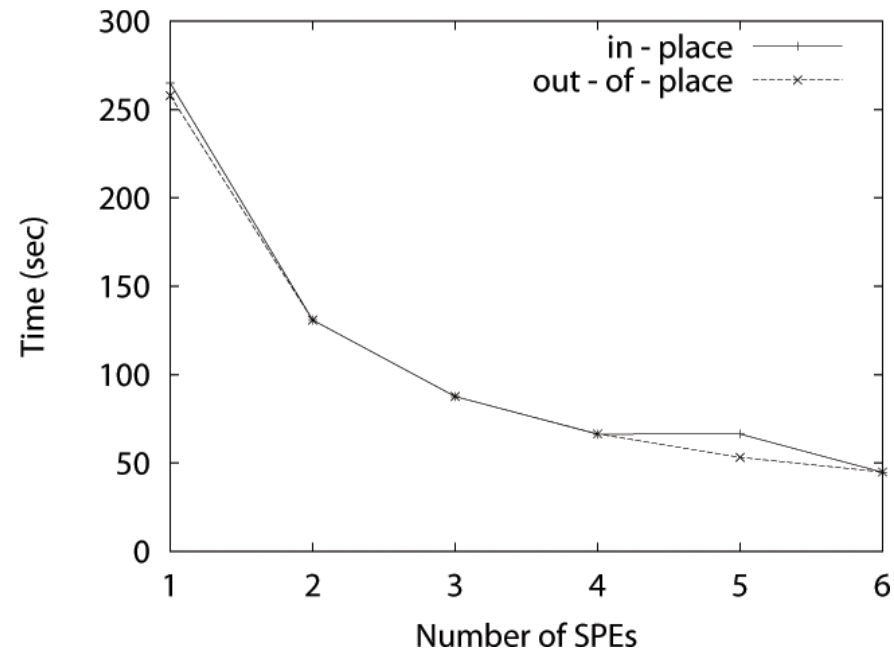


Convergence Time - Steps

Grid Size	Steps (iterations) to converge	
	In-place	Out-of-place
512 x 512	1305	2232
1024 x 1024	2340	4410
2048 x 2048	4455	8595
3072 x 3072	6570	12735
4096 x 4096	8685	16875
6144 x 6144	12870	25155

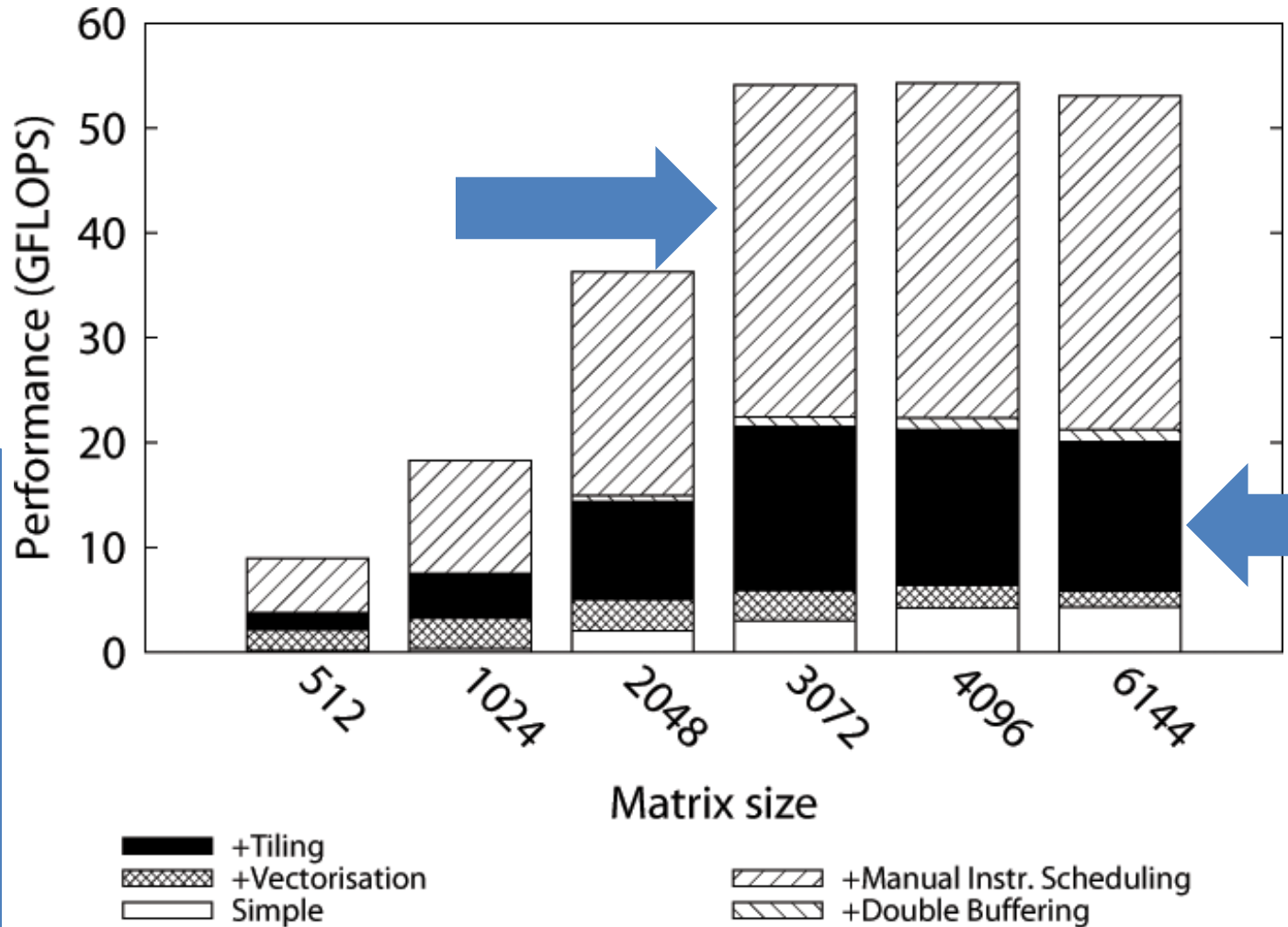
- In-place algorithm runs approximately twice as fast as out-of-place
- Total execution time between the two algorithms is almost the same

- Out-of-place algorithm takes about twice as many steps to reach the converged solution point compared to in-place

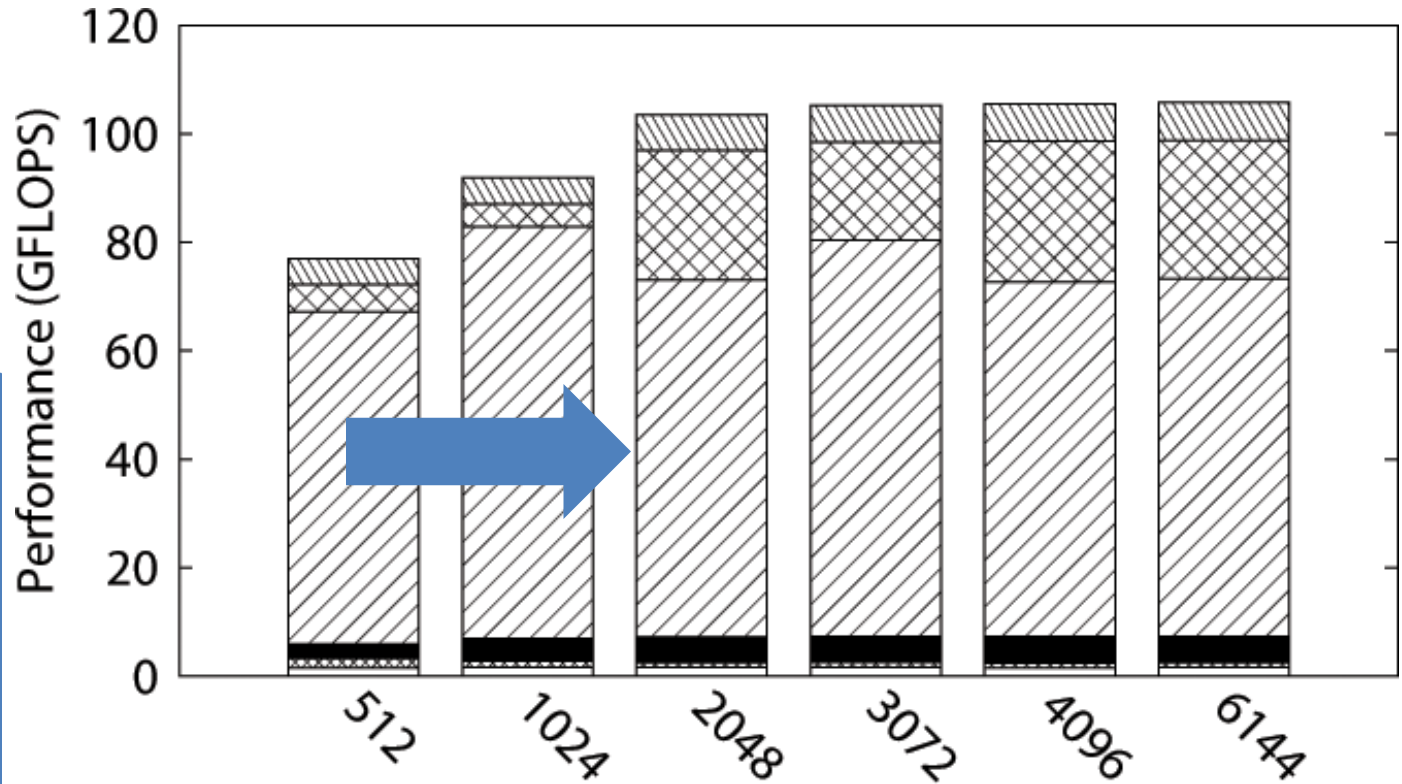


In-place performance improvements

➔ In the presence of all other optimizations, manual instruction scheduling almost doubles performance




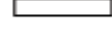


Out-of-place performance improvements



➔ Manual instruction scheduling still a determining factor; better scheduling opportunities

➔ Block-major layout prevents EIB congestion

 +Double Buffering
 +Tiling
 +Vectorization
 Simple

 +Quad Buffering
 +Block Major Layout
 +Manual Instr. Scheduling

Conclusions

- Overall execution time of both algorithms is similar, in-place being marginally faster
 - Out-of place is simpler to implement
 - In-place can be improved further by extending computations to more than one time steps concurrently (but code starts becoming overly complex)
- Taking advantage of as many architectural characteristics as possible plays important role
- But so does programmability
 - Tradeoff between performance and ease of programming
- ➡ Numerical criteria cannot be the sole factor when choosing an algorithm

Conclusions

- Block-major layout technique can reduce communication overhead; prevents EIB congestion
- Diagonal traversal proved to be a key point in vectorizing the in-place solver
- Producing code capable of fully exploiting the heterogeneous pipelines is the most significant factor in achieving high performance
 - Compiler optimizations alone yield performance far below the potential peak
 - Manual code optimizations (esp. instruction scheduling) is time-consuming

Future Work

- Implementation of same application on GPGPU platforms
- Three-dimensional advection PDE
- Other PDEs
- Other numerical schemes (e.g. multi-coloring schemes like Red-Black)
- Techniques to achieve better automatic instruction scheduling – research on compilers

- Questions?

{grokos, gpeteinatos, gkouv, goumas, kkourt, nkoziris}@cslab.ece.ntua.gr



Thank You