Software Engineering Seminar

Sebastian Hafen

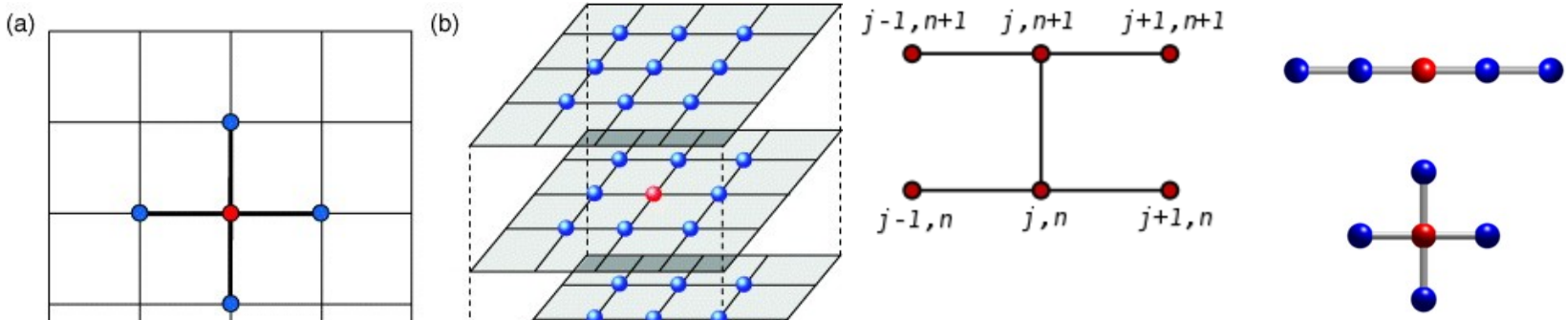
An Auto-Tuning Framework for Parallel Multicore Stencil Computations

Shoaib Kamil , Cy Chan , Leonid Oliker , John Shalf , Samuel Williams

Stencils

What is a Stencil Computation?

- Nearest Neighbor Computations
  - E.g. finite difference between data points
- Sweeps over a structured Grid
  - Like a n-dimensional Array
  - Iterative: i → i+1 → i+2

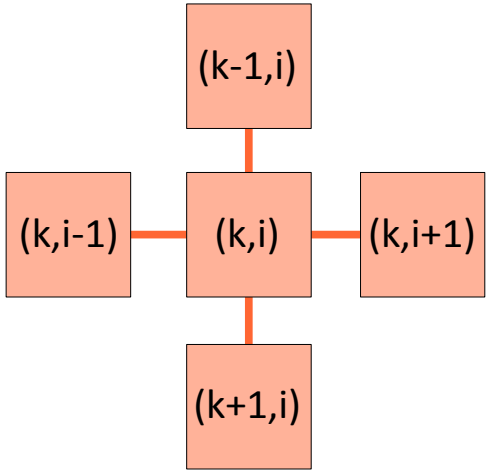


Left Two: http://iopscience.iop.org/1749-4699/2/1/015005/fulltext
Middle: http://en.wikipedia.org/wiki/Stencil_(numerical_analysis)
Right: http://en.wikipedia.org/wiki/Five-point_stencil

Example: 2D 5-Points-Stencil

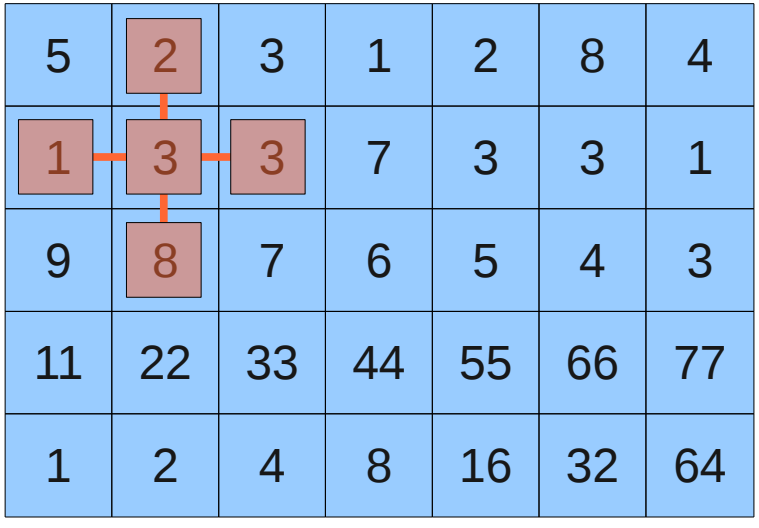
```
//Stencil-loop
do k=2, xLength-1, 1
  do i=2, yLength-1, 1
    writeArray[k][i] = useStencil(k,i)
  enddo
enddo
```

```
//Stencil-function
function useStencil(k,i)
  int result = readArray[k][i]
         + readArray[k+1][i]
         + readArray[k-1][i]
         + readArray[k][i+1]
         + readArray[k][i-1]
  result = result/5
  return result
endfunction
```

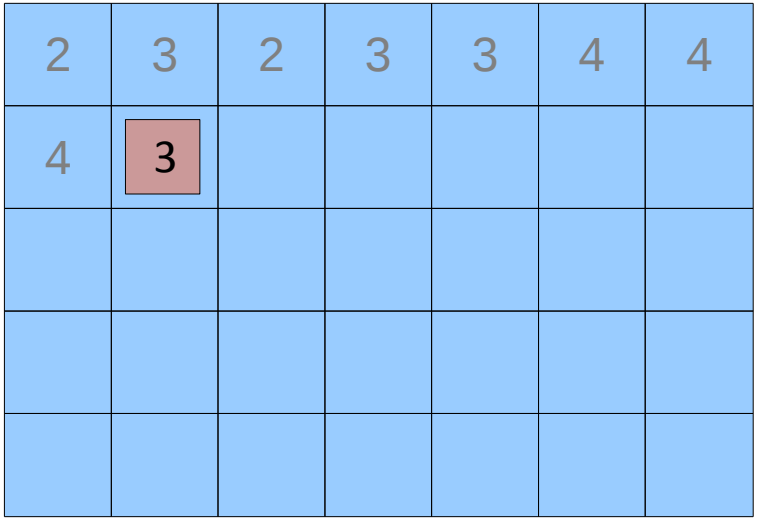


Example

readArray



writeArray



(2+1+3+3+8)/5 = 3

Example

readArray

5	2	3	1	2	8	4
1	3	3	7	3	3	1
9	8	7	6	5	4	3
11	22	33	44	55	66	77
1	2	4	8	16	32	64

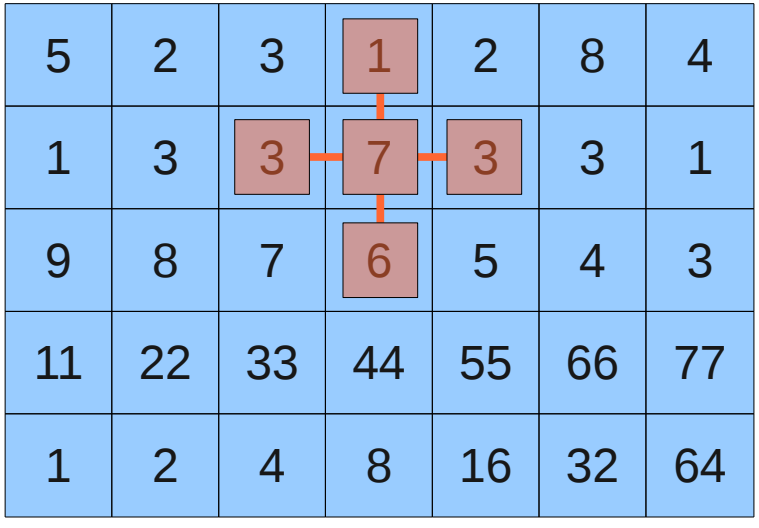
writeArray

2	3	2	3	3	4	4
4	3	4				

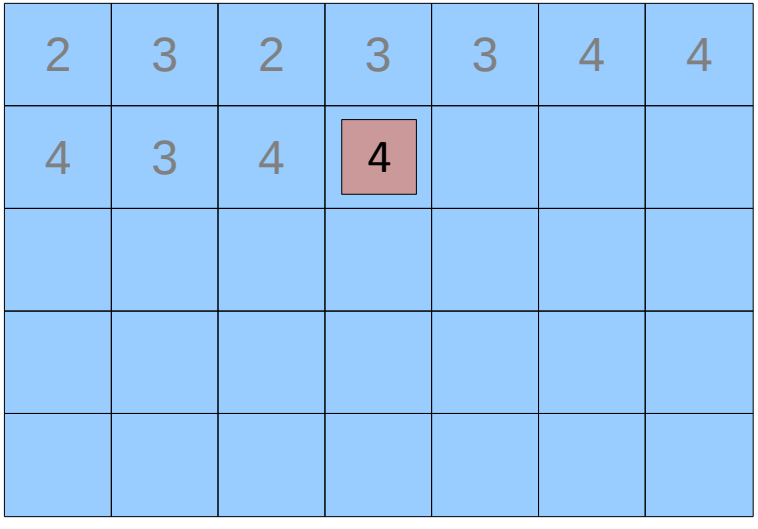
(3+3+3+7+7)/5 = 4

Example

readArray

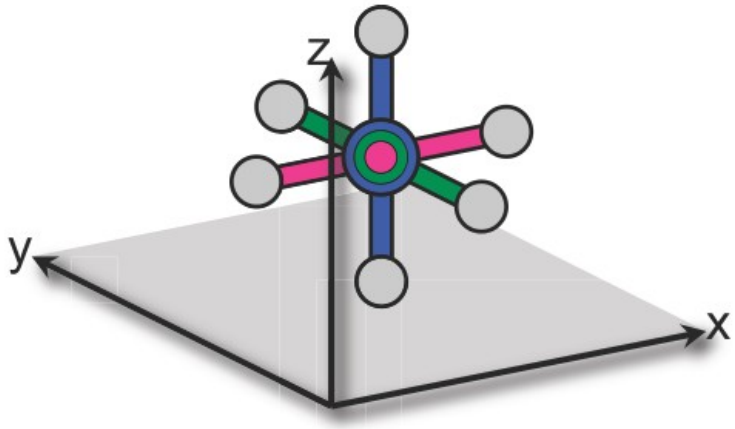


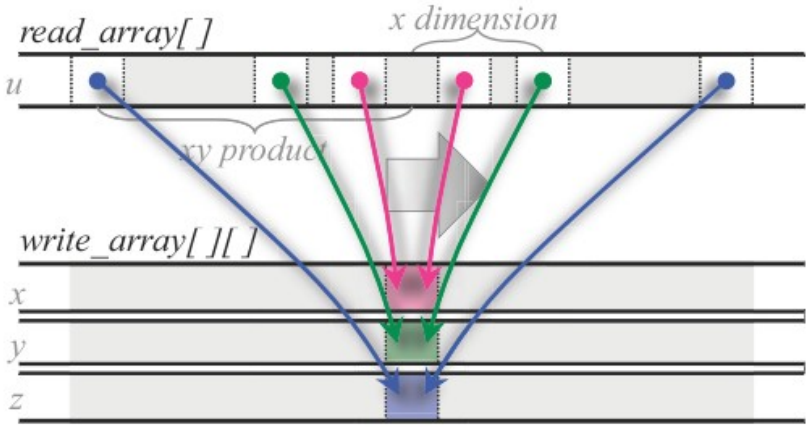
writeArray



(1+3+7+3+6)/5 = 4

Example from the paper: Gradient ∇



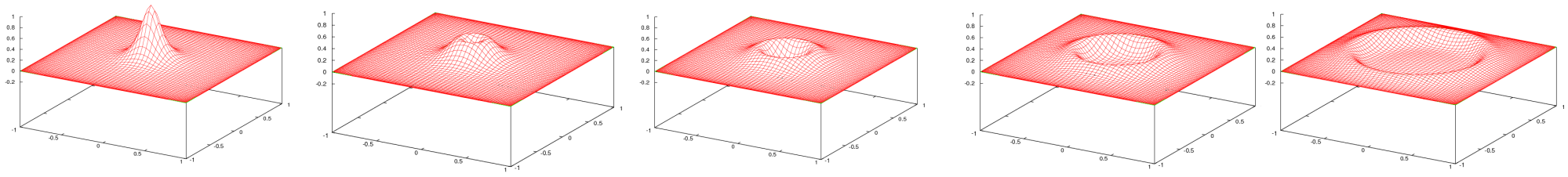



```
do k=2,nz-1,1
do j=2,ny-1,1
do i=2,nx-1,1

 x(i,j,k)=alpha*( u(i+1,j,k)-u(i-1,j,k) )
 y(i,j,k)= beta*( u(i,j+1,k)-u(i,j-1,k) )
 z(i,j,k)=gamma*( u(i,j,k+1)-u(i,j,k-1) )

enddo
enddo
enddo
```


Picture from Paper

Why?

- Solving Partial Differential Equations
  - Used by many branches of Science
    - Heat Equations
    - Wave Equations
    - “Automatic beam path analysis of laser wakefield particle acceleration data”
    - …



Quote: Papername of http://iopscience.iop.org/1749-4699/2/1/015005/fulltext
Images: http://www.math.uwaterloo.ca/~fpoulin/Files_html/fpcmresearch.html

Characteristics of stencil computations

- High memory traffic
- Low arithmetic intensity
  - CPUs can handle it

```
//Stencil-function
function useStencil(k,i)
  int result = readArray[k][i]
         + readArray[k+1][i]
         + readArray[k-1][i]
         + readArray[k][i+1]
         + readArray[k][i-1]
  result = result/5
  return result
endfunction
```

➔ Computations are memory bound

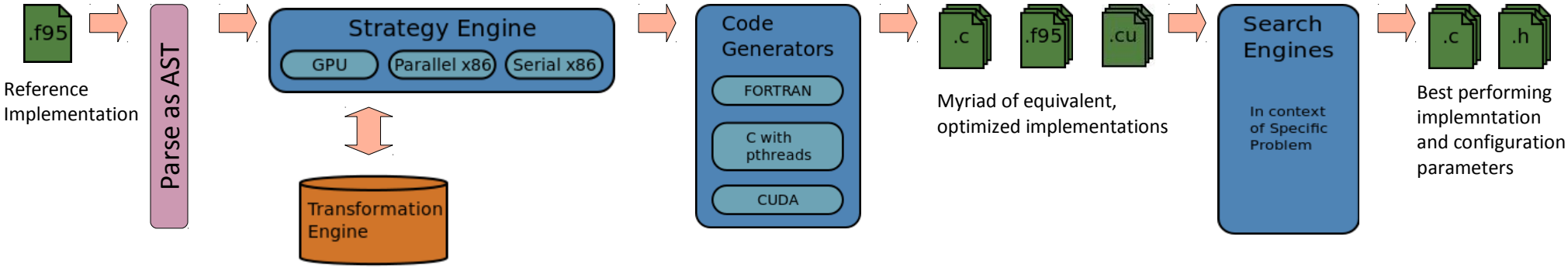
- Auto-tuning for better memory access management

The Framework

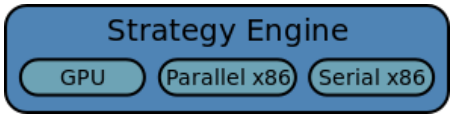
Overview

- Not the first auto-tuning framework for stencils
  - But other work about static/single kernel instantiations
- Proof-of-Concept
  - Supports broad range of stencil kernels
    - Fully generalized framework
  - Auto-parallelisation
  - Multiple back-end architectures
    - Even a GPU

Framework flow



Inspired by a picture of the paper



Strategy Engine

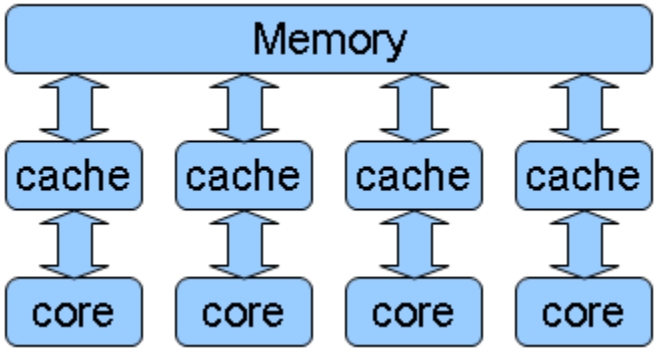
- Parameter Space is massive
  - Combined serial and parallel optimizations
- Decides on a appropriate subset of parameter combinations (strategies)
  - Based on the underlying architecture
- Knows about correlation of different optimizations
  - Chooses only legal combinations

Transformation Engine

- Transforms the AST
  - First applies auto-parallelization
  - Then uses auto-tuning
- Has domain knowledge
  - Can do transformations a compiler can not

Auto-parallelization

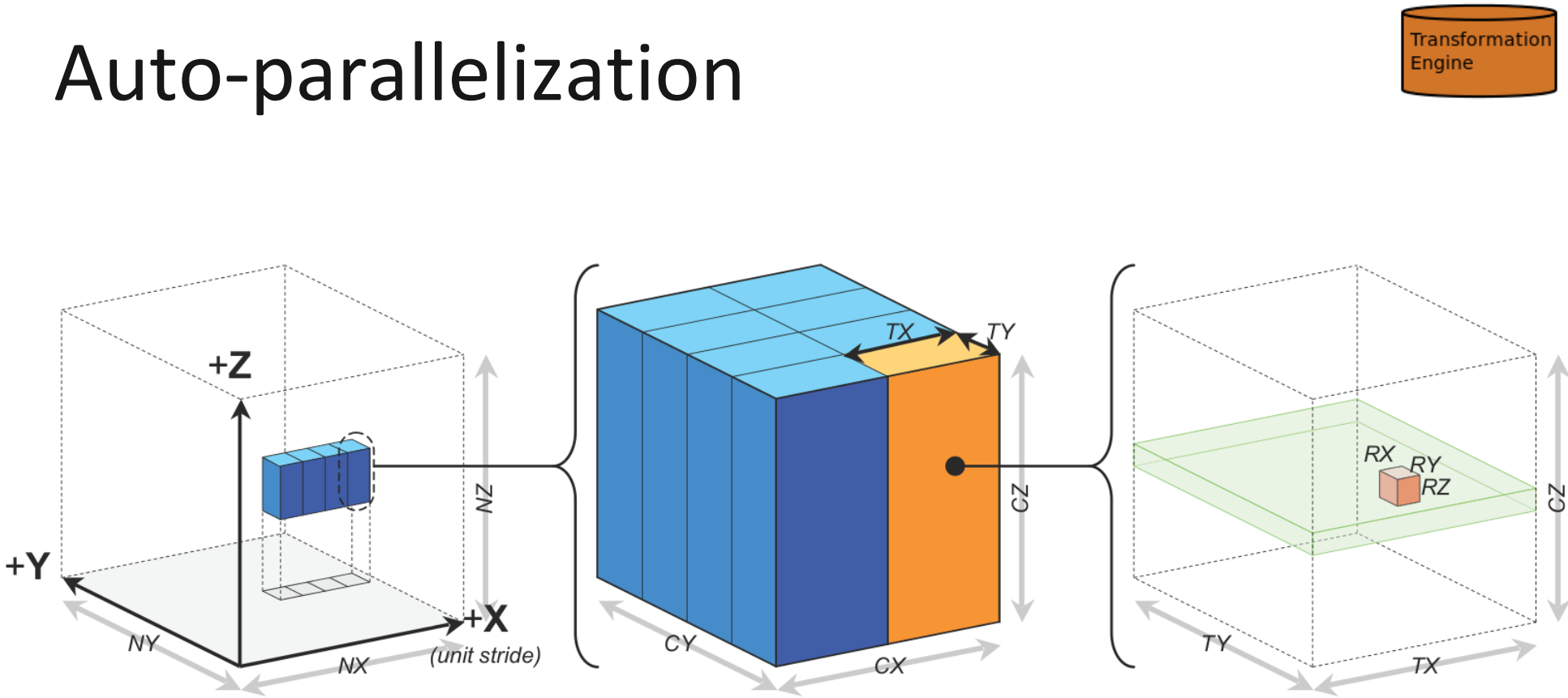
- Basically dividing the problem space into blocks
  - Core blocks, thread blocks and register blocks
  - Creates new loops for every block
- Non-Uniform Memory Access (NUMA)-Aware



- Separate stencil for the border cases

Image: http://www.1024cores.net/home/parallel-computing/cache-oblivious-algorithms

Auto-parallelization



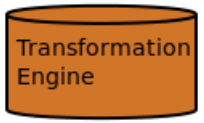
(a)
Decomposition of a Node Block into a Chunk of Core Blocks

(b)
Decomposition into Thread Blocks

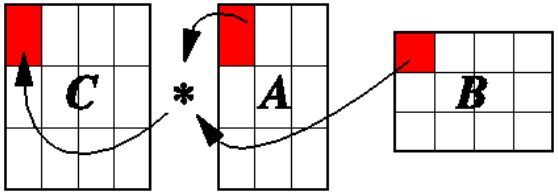
(c)
Decomposition into Register Blocks

Picture from Paper

Auto-tuning



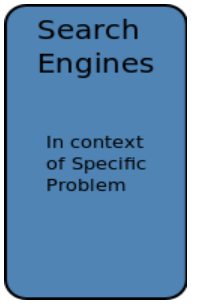
- Loop unrolling and register blocking
  - Improves innermost loop efficiency
- Cache blocking
  - Exposes temporal locality and and increases cache reuse



- Arithmetic simplifications
- Many more possible
  - It is a prove-of-concept

Example for cache blocking: http://techpubs.sgi.com/library/dynaweb_docs/0640/SGI_Developer/books/OrOn2_PfTune/sgi_html/ch06.html

Search Engine

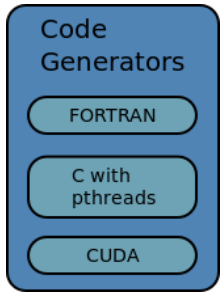


- Runs all the different tuned versions of the stencil kernel
  - 256^3 grids (16'777'216 Elements) initialized with random values
- User can replace the original kernel with the fastest one

Limitations

- Only 2D or 3D
- Only Arrays
  - No sophisticated Data structures
- Only arithmetic stencils
- They want to change that in future work

Code Generator

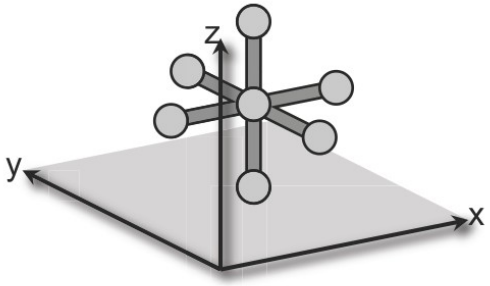


- Creates code from the modified ASTs
  - For the CPUs: pthreads
  - For the GPU: CUDA thread blocks
  - Serial fortran and c code also possible

Tested
Stencils and Architectures

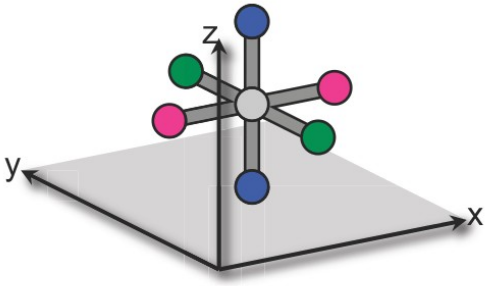
Used Stencils

Laplacian Stencil



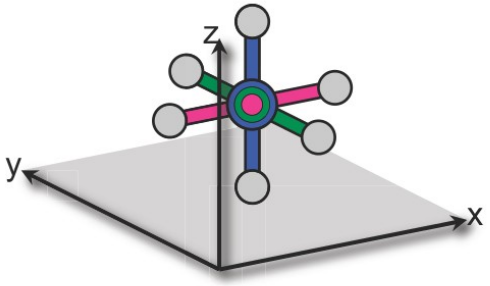

```
do k=2,nz-1,1
do j=2,ny-1,1
do i=2,nx-1,1

 uNext(i,j,k)=
  alpha*u(i,j,k)+
  beta*(u(i+1,j,k)+u(i-1,j,k)+
        u(i,j+1,k)+u(i,j-1,k)+
        u(i,j,k+1)+u(i,j,k-1)
       )
enddo
enddo
enddo
```

Divergence Stencil

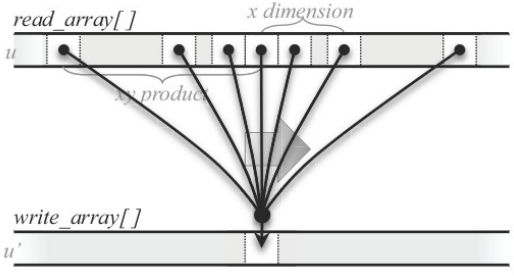


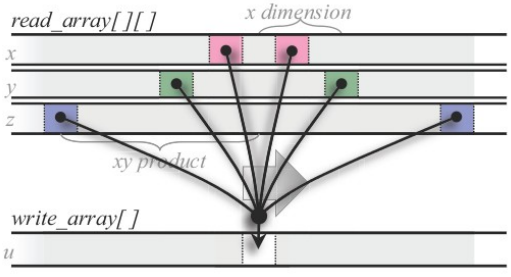

```
do k=2,nz-1,1
do j=2,ny-1,1
do i=2,nx-1,1

 u(i,j,k)=
  alpha*( x(i+1,j,k)-x(i-1,j,k) )+
   beta*( y(i,j+1,k)-y(i,j-1,k) )+
  gamma*( z(i,j,k+1)-z(i,j,k-1) )

enddo
enddo
enddo
```

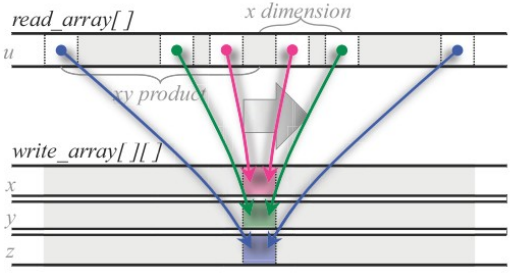
Gradient Stencil




```
do k=2,nz-1,1
do j=2,ny-1,1
do i=2,nx-1,1

 x(i,j,k)=alpha*( u(i+1,j,k)-u(i-1,j,k) )
 y(i,j,k)= beta*( u(i,j+1,k)-u(i,j-1,k) )
 z(i,j,k)=gamma*( u(i,j,k+1)-u(i,j,k-1) )

enddo
enddo
enddo
```







Picture from Paper

Used Architectures

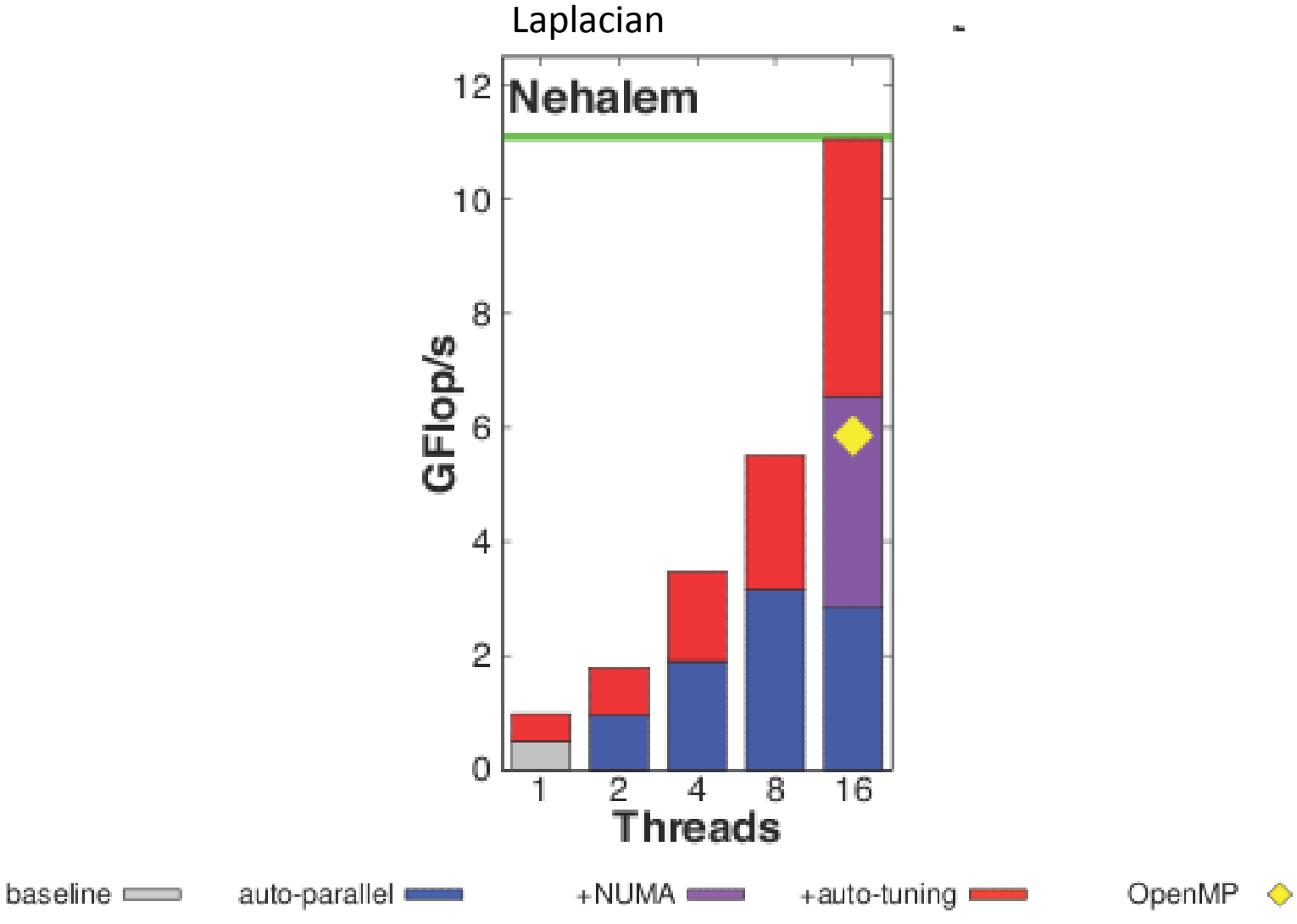
Core Architecture	AMD Barcelona	Intel Nehalem	Sun Niagara2
Type	superscalar out of order	superscalar out of order	HW multithread dual issue
Clock (GHz)	2.30	2.66	1.16
DP GFlop/s	9.2	10.7	1.16
Local-Store	—	—	—
L1 Data Cache	64KB	32KB	8KB
private L2 cache	512KB	256KB	—

System Architecture	Opteron 2356 (Barcelona)	Xeon X5550 (Gainestown)	UltraSparc T5140 (Victoria Falls)
# Sockets	2	2	2
Cores per Socket	4	4	8
Threads per Socket‡	4	8	64
primary memory parallelism paradigm	HW prefetch	HW prefetch	Multithreading
shared L3 cache	2×2MB (shared by 4 cores)	2×8MB (shared by 4 cores)	2×4MB (shared by 8 cores)
DRAM Capacity	16GB	12GB	32GB
DRAM Pin Bandwidth (GB/s)	21.33	51.2	42.66(read) 21.33(write)
DP GFlop/s	73.6	85.3	18.7
DP Flop:Byte Ratio	3.45	1.66	0.29
Threading	Pthreads	Pthreads	Pthreads
Compiler	gcc 4.1.2	gcc 4.3.2	gcc 4.2.0

Picture from Paper

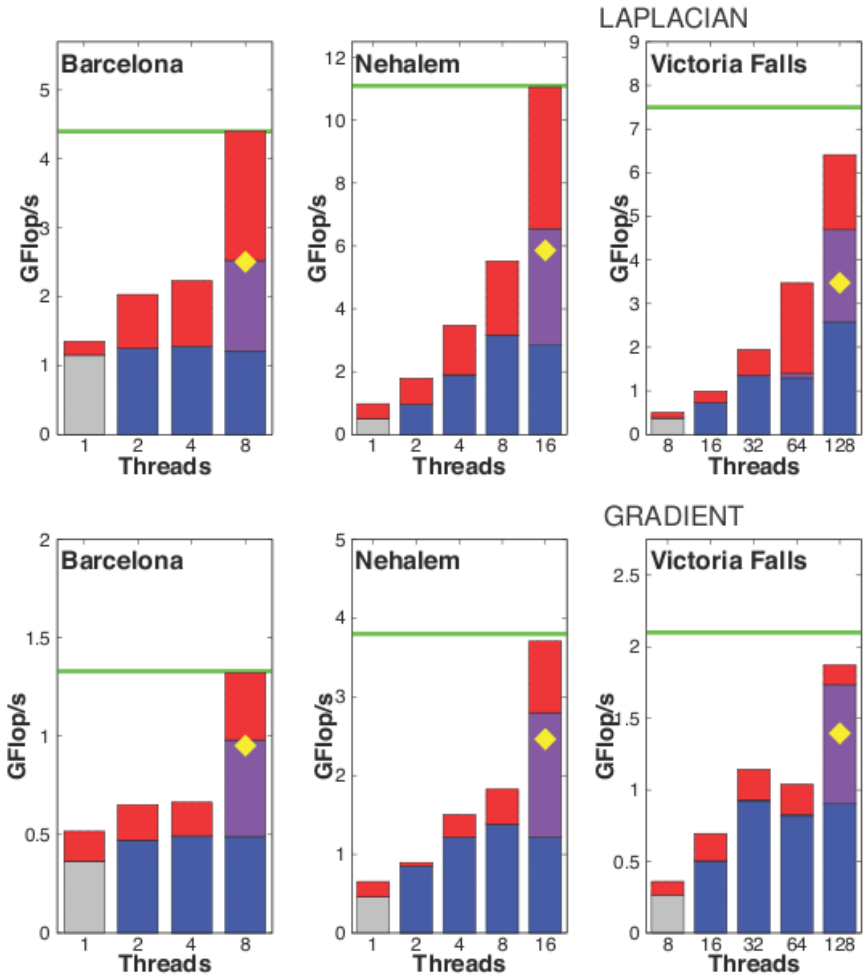
Results

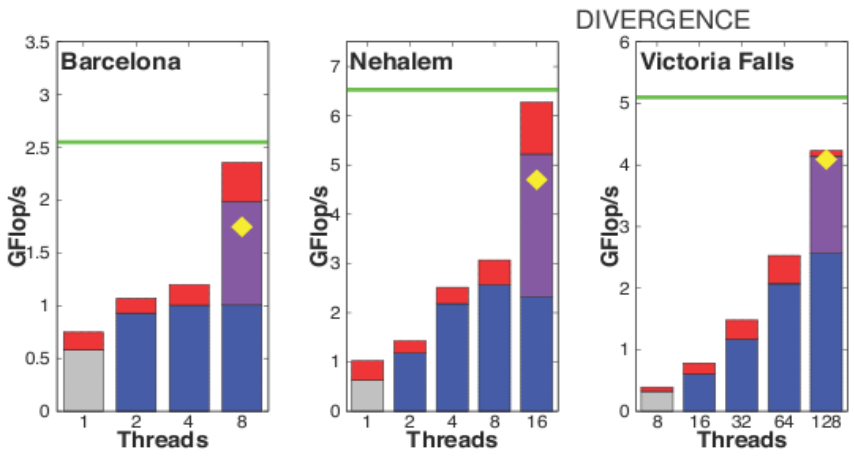
One Result



Pictures from Paper

Results





baseline auto-parallel +NUMA +auto-tuning OpenMP

Pictures from Paper

Conclusion

- Pro
  - It does work. Concept is proven
    - Fully general
  - Performance comparable to hand-optimized code
  - “Programmer Production Benefits”
    - Few minutes to annotate code
- Contra
  - OpenMP works good, too
  - New architecture means new coding
  - Peak not yet reached

Quote from Paper

End of Presentation