Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures

Kaushik Datta*†, Mark Murphy†, Vasily Volkov†, Samuel Williams*†, Jonathan Carter*, Leonid Oliker*†, David Patterson*†, John Shalf*, and Katherine Yelick*†

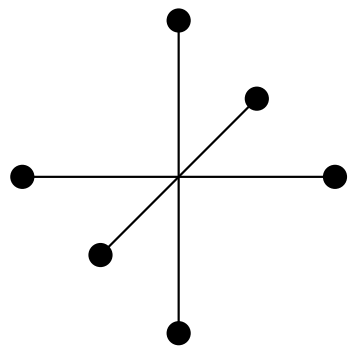
*CRD/NERSC, Lawrence Berkeley National Laboratory
†Computer Science Division, University of California at Berkeley

Publication Date: 2008

Presented by: Lukas Spies, 2018-12-05

Motivation:

- Stencils commonly used in Scientific Computing Applications
- challenge: small re-use of data
- common approach: tiling
- this paper: study of various optimizations and architectures
- Sample stencil: 7-point stencil (representative)

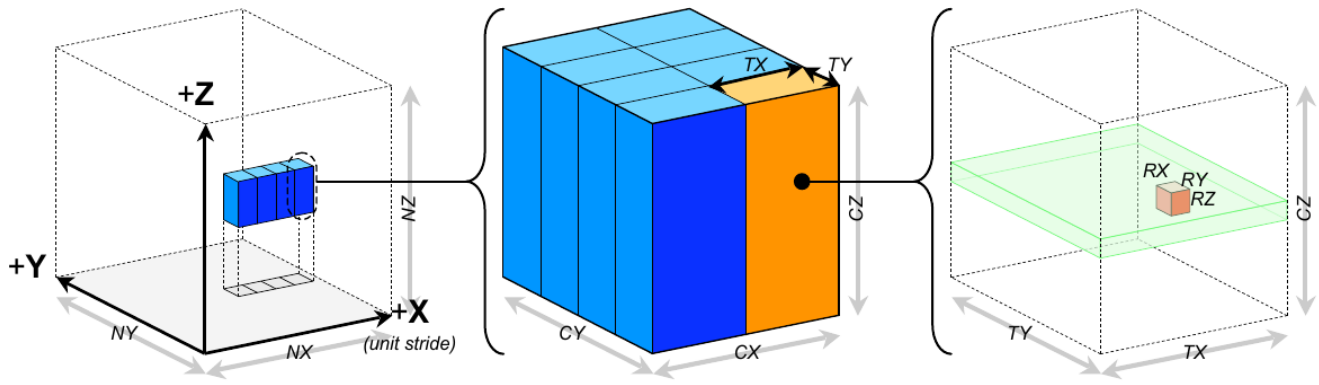


Core Architecture	Intel Core2	AMD Barcelona	Sun Niagara2	STI Cell eDP SPE	NVIDIA GT200 SM
Type	super scalar out of order	super scalar out of order	MT dual issue†	SIMD dual issue	MT SIMD
Process	65nm	65nm	65nm	65nm	65nm
Clock (GHz)	2.66	2.30	1.16	3.20	1.3
DP GFlop/s	10.7	9.2	1.16	12.8	2.6
Local-Store	—	—	—	256KB	16KB**
L1 Data Cache	32KB	64KB	8KB	—	—
private L2 cache	—	512KB	—	—	—

System	Xeon E5355 (Clovertown)	Opteron 2356 (Barcelona)	UltraSparc T5140 T2+ (Victoria Falls)	QS22 PowerXCell 8i (Cell Blade)	GeForce GTX280
Heterogeneous	no	no	no	multicore	multichip
# Sockets	2	2	2	2	1
Cores per Socket	4	4	8	8(+1)	30 (×8)
shared L2/L3 cache	4×4MB (shared by 2)	2×2MB (shared by 4)	2×4MB (shared by 8)	—	—
DP GFlop/s	85.3	73.6	18.7	204.8	78
primary memory parallelism paradigm	HW prefetch	HW prefetch	Multithreading	DMA	Multithreading with coalescing
DRAM Bandwidth (GB/s)	21.33(read) 10.66(write)	21.33	42.66(read) 21.33(write)	51.2	141 (device) 4 (PCIe)
DP Flop:Byte Ratio	2.66	3.45	0.29	4.00	0.55
DRAM Capacity	16GB	16GB	32GB	32GB	1GB (device) 4GB (host)
System Power (Watts)§	330	350	610	270‡	450 (236)⋆
Chip Power (Watts)¶	2×120	2×95	2×84	2×90	165
Threading	Pthreads	Pthreads	Pthreads	libspe2.1	CUDA 2.0
Compiler	icc 10.0	gcc 4.1.2	gcc 4.0.4	xlc 8.2	nvcc 0.2.1221

Optimization #1:

Problem Decomposition:
Division of work into node blocks, core blocks, thread blocks and register blocks



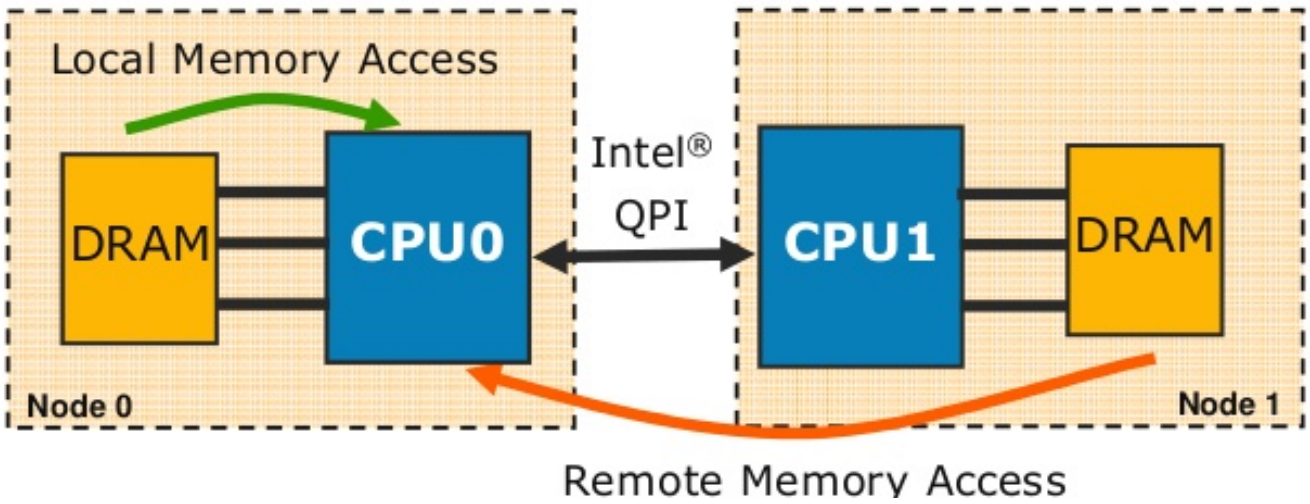
(a)
Decomposition of a Node Block into a Chunk of Core Blocks

(b)
Decomposition into Thread Blocks

(c)
Decomposition into Register Blocks

Optimization #2:

Data Allocation:
NUMA's "first touch" page mapping

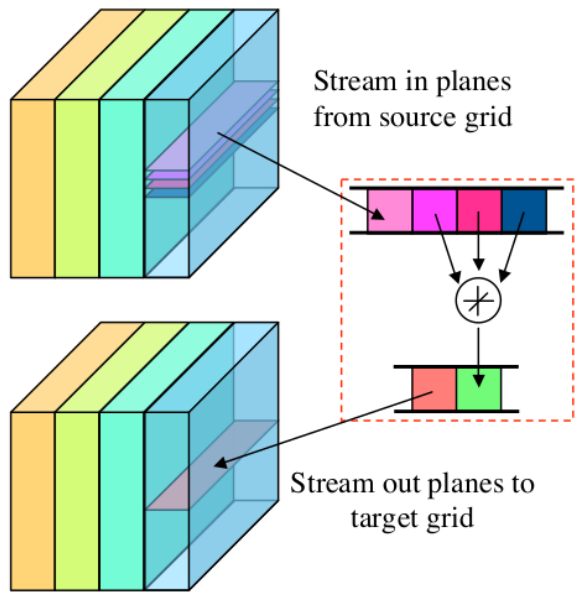


- remote memory latency $>$ local memory latency
- remote memory bandwidth $<$ local memory bandwidth

Optimization #3:

Bandwidth Optimizations:

- hide memory latency
- minimize memory traffic



Circular queue optimization:

planes are streamed into a queue containing the current time step, processed, written to out queue, and streamed back → minimizes memory traffic

Optimization #4:

In-Core Optimizations:

- inner loop transformations
- handwritten SIMD code

```
// N = 4*512;
for(int i = 0; i < N; ++i) {
    val[i] = in1[i]*in2[i];
}

// when unrolled manually becomes

for(int i = 0; i < N/4; ++i) {
    val[4*i   ] = in1[4*i   ]*in2[4*i   ];
    val[4*i +1] = in1[4*i +1]*in2[4*i +1];
    val[4*i +2] = in1[4*i +2]*in2[4*i +2];
    val[4*i +3] = in1[4*i +3]*in2[4*i +3];
}
```

Auto-tuner:

- Not many details provided
- Consisting of two components:
  1. Perl code generator producing multithreaded C code: allows evaluation of a large optimization space
  2. auto-tuning benchmark that searches parameter space

Category	Optimization Parameter	Name	parameter tuning range by architecture: Clovertown	Barcelona	Victoria Falls	Cell Blade	GTX280
Data Allocation	NUMA Aware		N/A	✓	✓	✓	N/A
	Pad to a multiple of:		1	1	1	16	16
Domain Decomp	Core Block Size	*CX*	NX	NX	{8...NX}	{64...NX}	{16...32}
		CY	{8...NY}	{8...NY}	{8...NY}	{8...NY}	CX
		CZ	{128...NZ}	{128...NZ}	{128...NZ}	{128...NZ}	64
	Thread Block Size	*TX*	CX	CX	{8...CX}	CX	1
		TY	CY	CY	{8...CY}	CY	CY/4
	Chunk Size		$\{1...\frac{NX \times NY \times NZ}{CX \times CY \times CZ \times NThreads}\}$				N/A
Low Level	Register Block Size	*RX*	{1...8}	{1...8}	{1...8}	2	TX
		RY	{1...2}	{1...2}	{1...2}	8	TY
		RZ	{1...2}	{1...2}	{1...2}	1	1
	(explicitly SIMDized)		✓	✓	N/A	✓	N/A
	Prefetching Distance		{0...64}	{0...64}	{0...64}	N/A	N/A
	DMA Size		N/A	N/A	N/A	CX×CY	N/A
	Cache Bypass		✓	✓	N/A	implicit	implicit
	Circular Queue		—	—	—	✓	✓

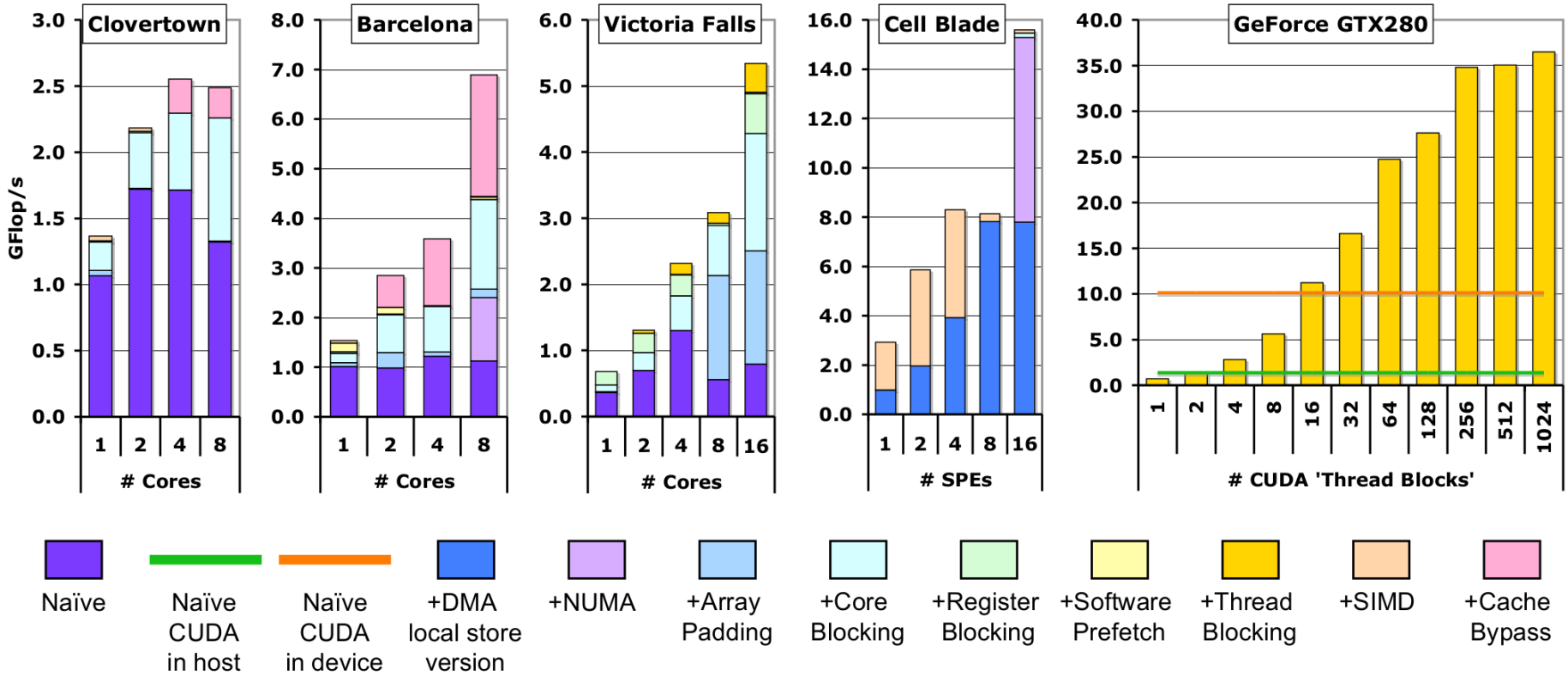
Table 2. Attempted optimizations and the associated parameter spaces explored by the auto-tuner for a 256^3 stencil problem ($NX, NY, NZ = 256$). All numbers are in terms of doubles.

Problem set up:

- Sample problem: 256^3 stencil calculation
  $\Rightarrow$ 262MB of memory
- Double precision
- not all techniques/optimizations available on every platform

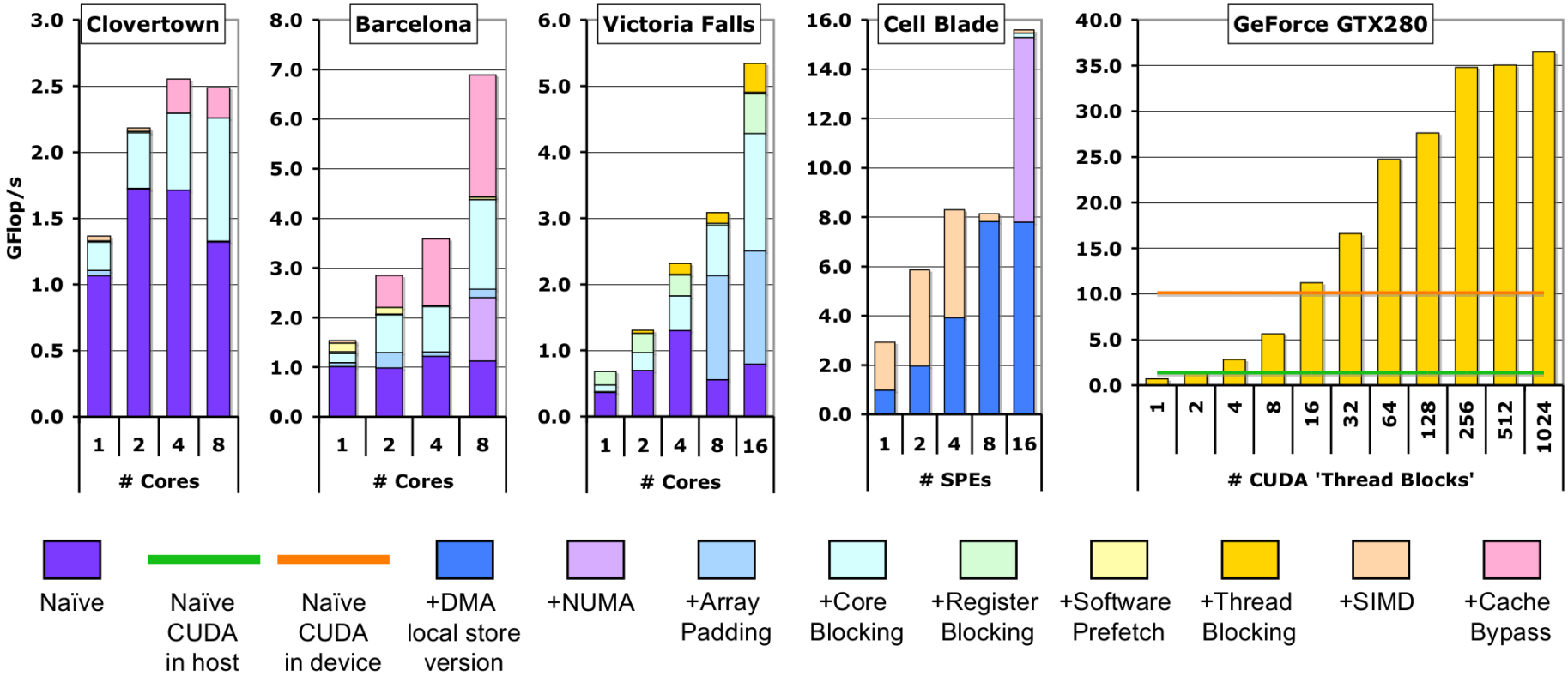
Details not provided/available:

- No code provided for any of the test run
- some of the specific optimizations are not described



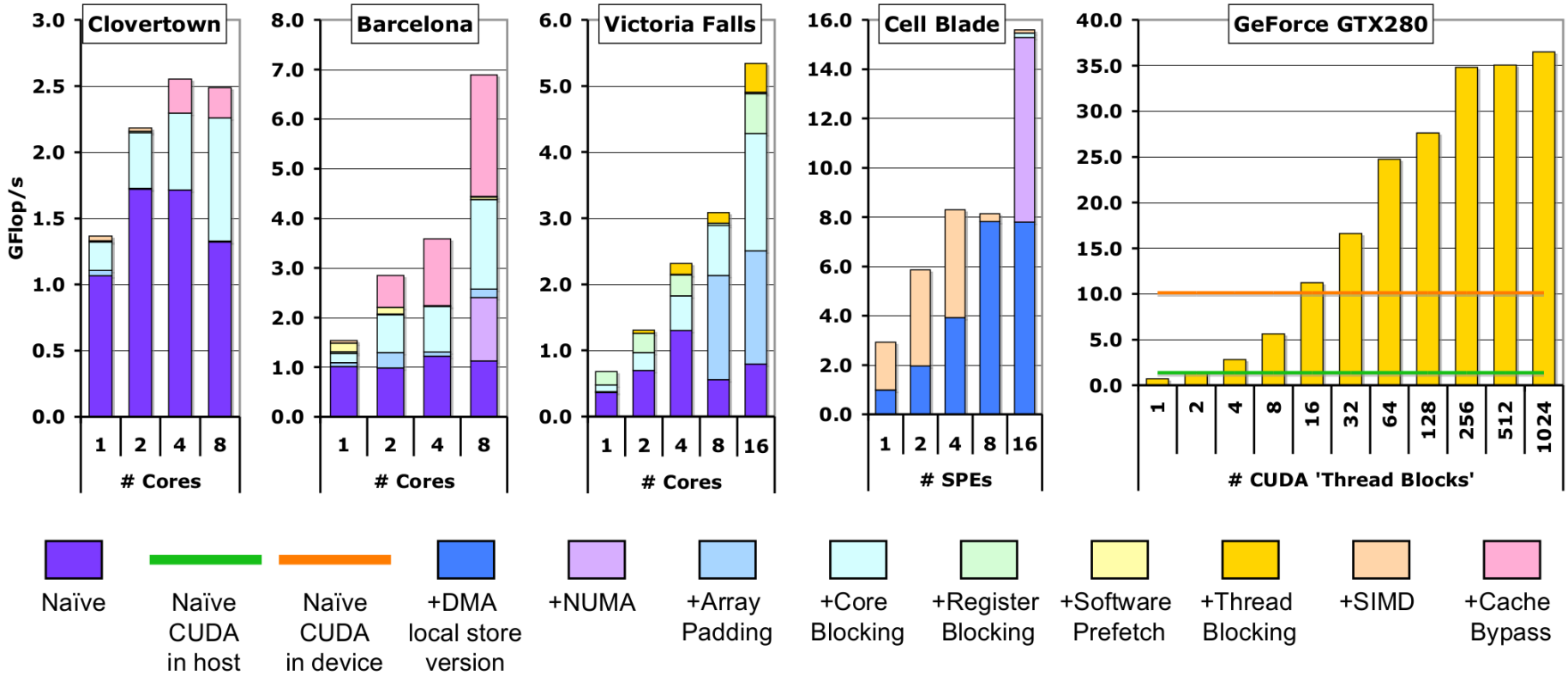
Clovertown

- notable performance benefits:
  core blocking (1.7), cache bypass (1.1)
- uniform memory access → NUMA optimizations no effect
- poor multicore scaling → memory bandwidth limited



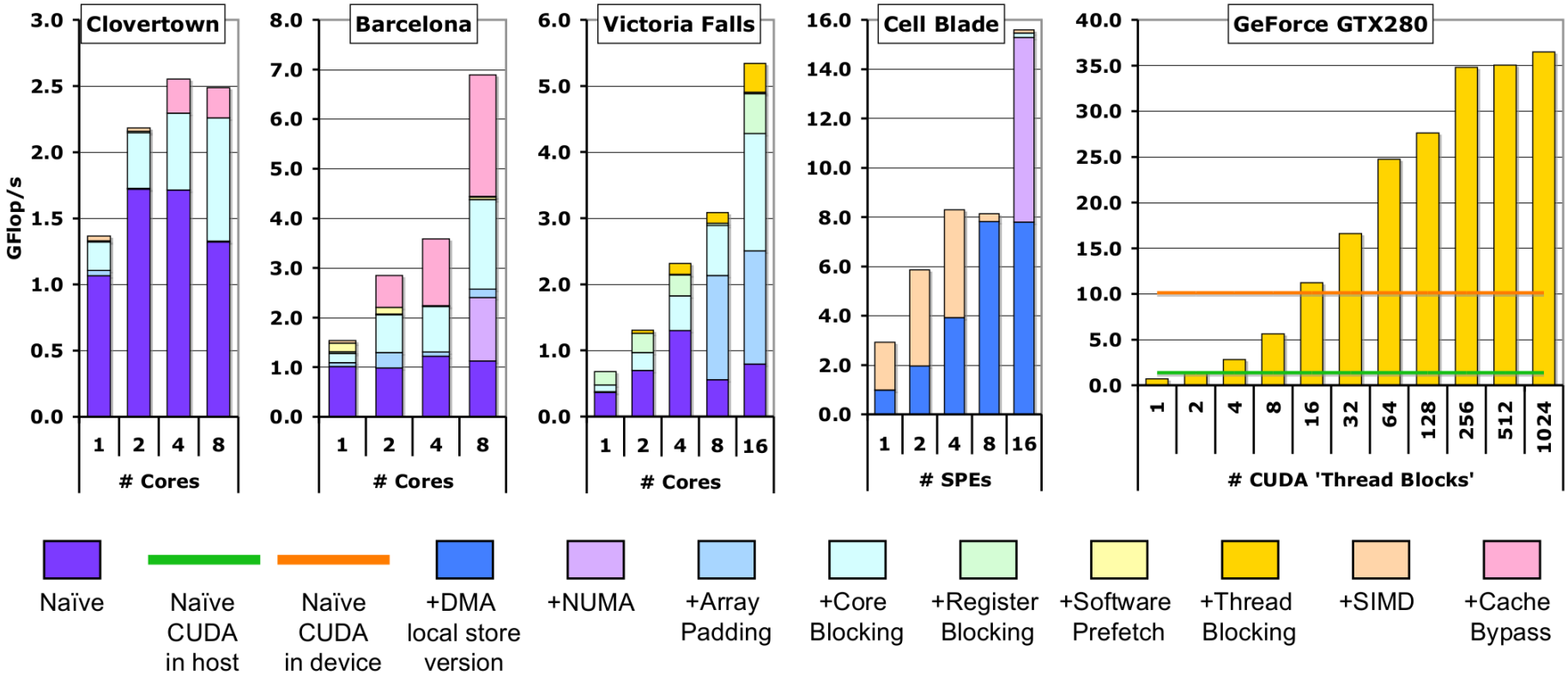
Barcelona

- notable performance benefits:
  NUMA optimizations (2.15), core-blocking (1.7), cache bypass (1.55)
- memory bandwidth limited
- register blocking, software prefetching little to no effect



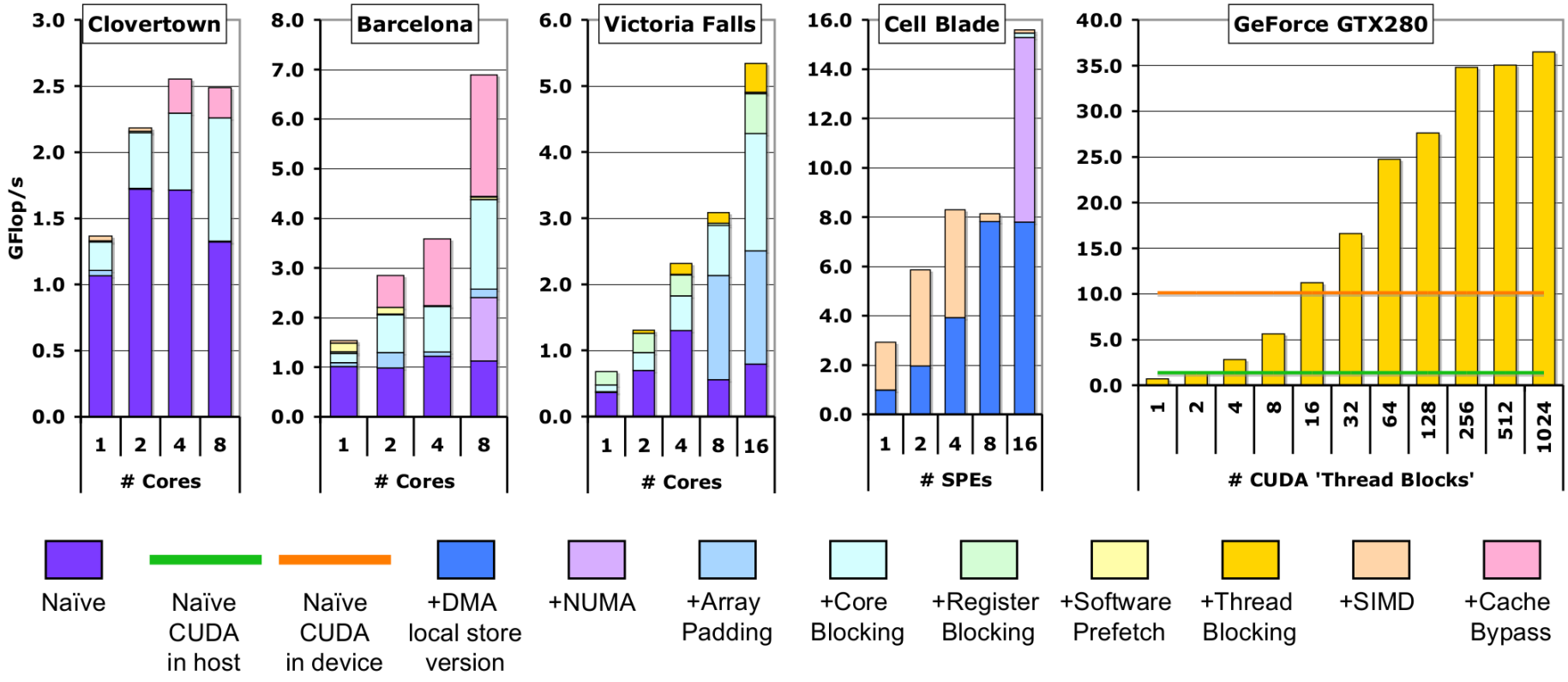
Victoria Falls

- notable performance benefits:
  array padding and core/register blocking (6.1), thread blocking (1.1)
- provides better per-core cache behavior
- large parameter search space (lengthy process)



Cell Blade

- generic microprocessor-targeted source code cannot be naively compiled → DMA local-store version as baseline
- computationally bound (1-4 cores) → SIMD
- memory bandwidth limited (8+ cores) → NUMA



GTX280

- naive version: one CUDA thread responsible for single calculations (one stencil)
- 2 categories: naive CUDA in host, naive CUDA in device
- bottleneck: PCIe x16 sustained bandwidth of only 3.4 GB/s
- low arithmetic intensity limiting factor

Conclusions:

- parallelism discovery only small part of performance challenge: hardware parallelism and memory hierarchy optimizations of equal importance
- large number of simpler processors (GTX280) offer higher performance potential than small number of more complex processors (Clovertown)
- novel strategies for hiding memory latency effective, but at expense of programming productivity
- GPU provides very good on-device performance, PCIe bandwidth bottleneck
- cache-based architectures show complete lack of multicore scalability without auto-tuning:
  naive implementations do not profit from more cores
- Compilers can do a lot of optimizations, but more is often possible