Optimizing and Tuning the Fast Multipole Method for Multicore and Accelerator Systems

Georgia Tech

– *Aparna Chandramowlishwaran,* Aashay Shringarpure, Ilya Lashuk; George Biros, Richard Vuduc

Lawrence Berkeley National Laboratory

– Sam Williams, Lenny Oliker

IPDPS 2010

Key Ideas and Findings

- First cross-platform single-node multicore study of tuning the fast multipole method (FMM)
  - Explores data structures, SIMD, multithreading, mixed-precision, and tuning
  - Show 25x speedups on Intel Nehalem, 9.4x AMD Barcelona, 37.6x Sun Victoria Falls
- **Surprise?** Multicore ~ GPU in performance & energy efficiency for the FMM
- Broader context: Generalized n-body problems, for particle simulation & statistical data analytics

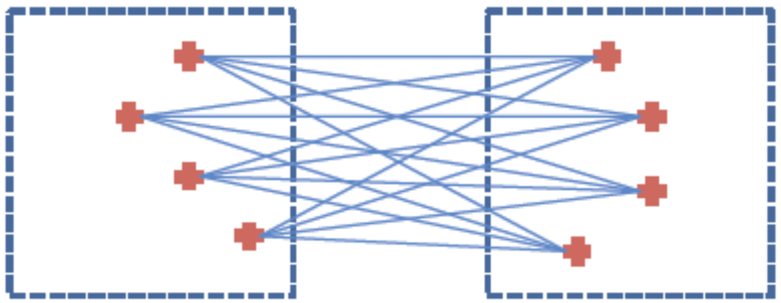
- High-performance multicore FMMs: Analysis, optimization, and tuning
  - Algorithmic characteristics
  - Architectural implications
  - Observations

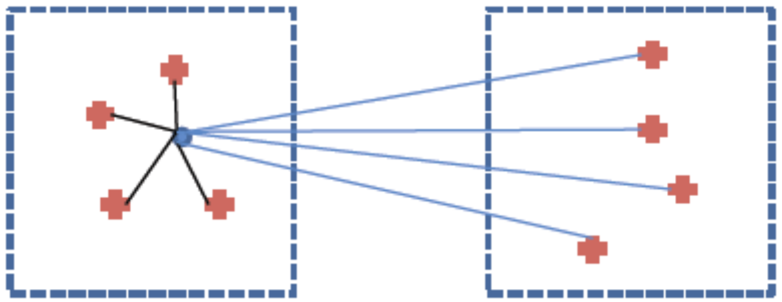
- High-performance multicore FMMs: Analysis, optimization, and tuning
  - **Algorithmic characteristics**
  - Architectural implications
  - Observations

Computing Direct vs. Tree-based Interactions

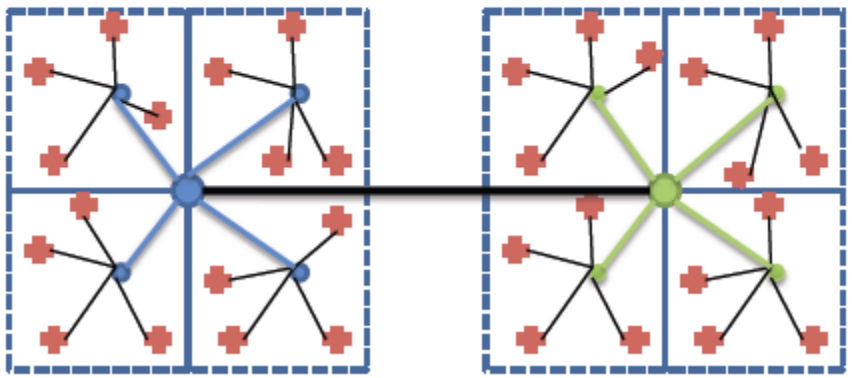
Direct evaluation: $O(N^2)$

Barnes-Hut: $O(N \log N)$



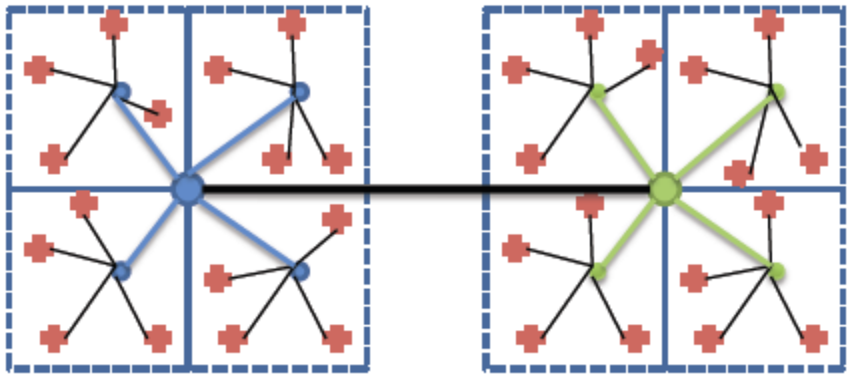


Fast Multipole Method (FMM): $O(N)$

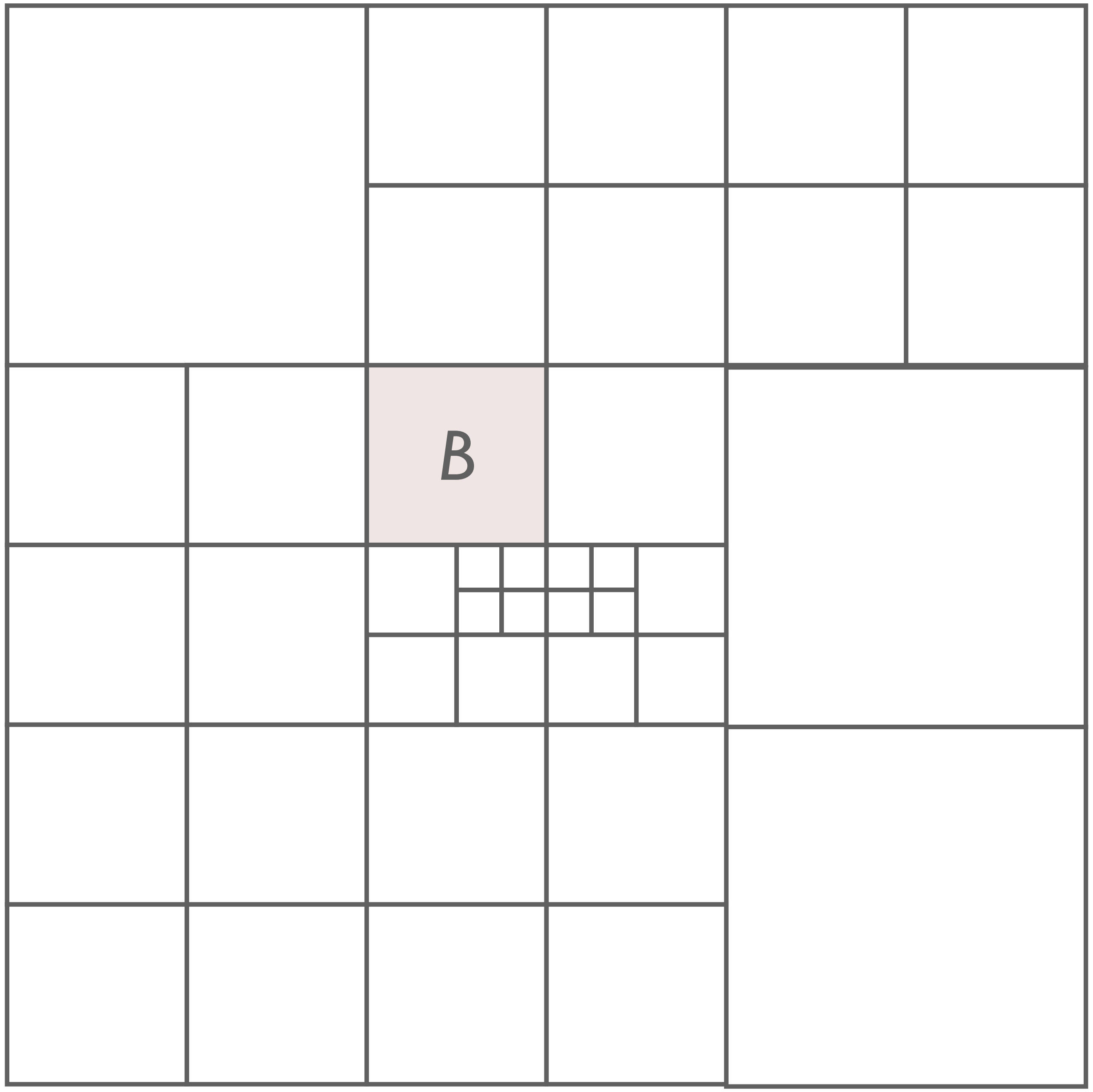


Fast multipole method

- Given:
  - *N* target points and *N* sources
  - Tree type & max points per leaf, *q*
  - Desired accuracy, ε
- Two steps
  - Build tree
  - Evaluate potential at all *N* targets

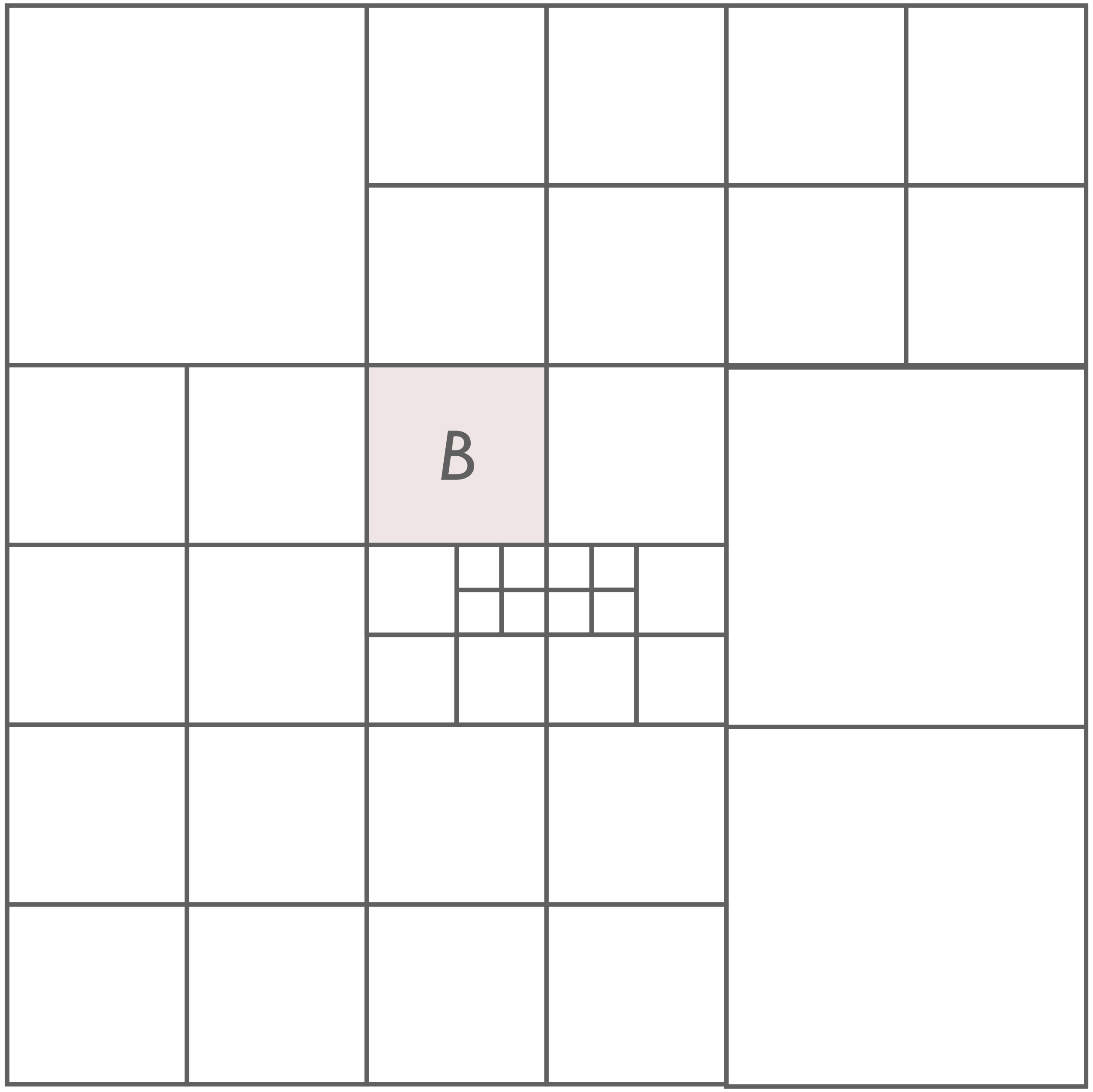


We use *kernel-independent FMM* (KIFMM) of Ying, Zorin, Biros (2004).



Tree construction

Recursively divide space until each box has **at most *q* points**.

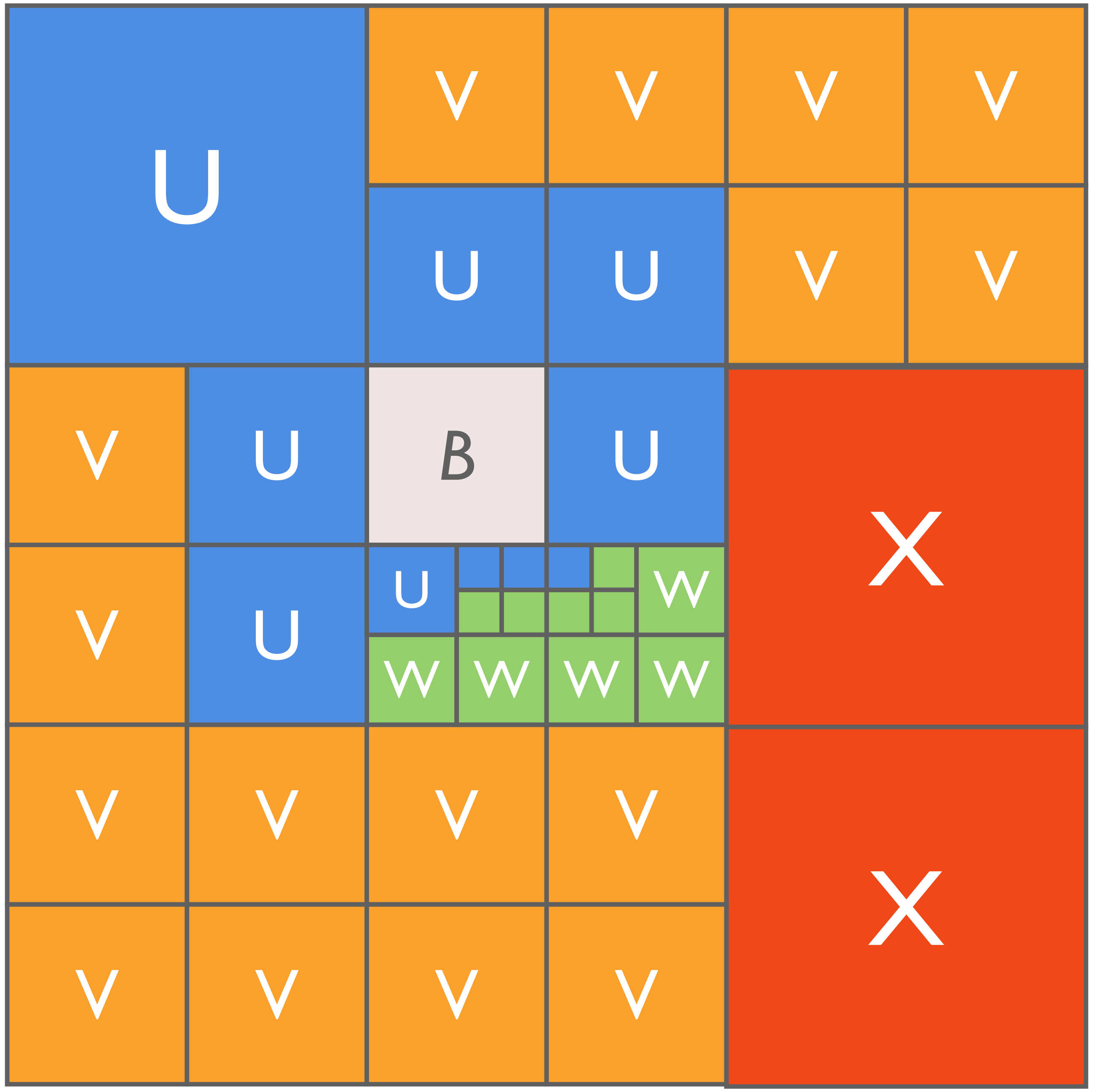


Six phases:
(1.) Upward pass
(2–5.) List computations
(6.) Downward pass

Phases vary in:
→ data parallelism
→ intensity (flops : mops)

Evaluation phase

Given the adaptive tree, FMM evaluation performs a series of tree traversals, doing some work at each node, *B*.

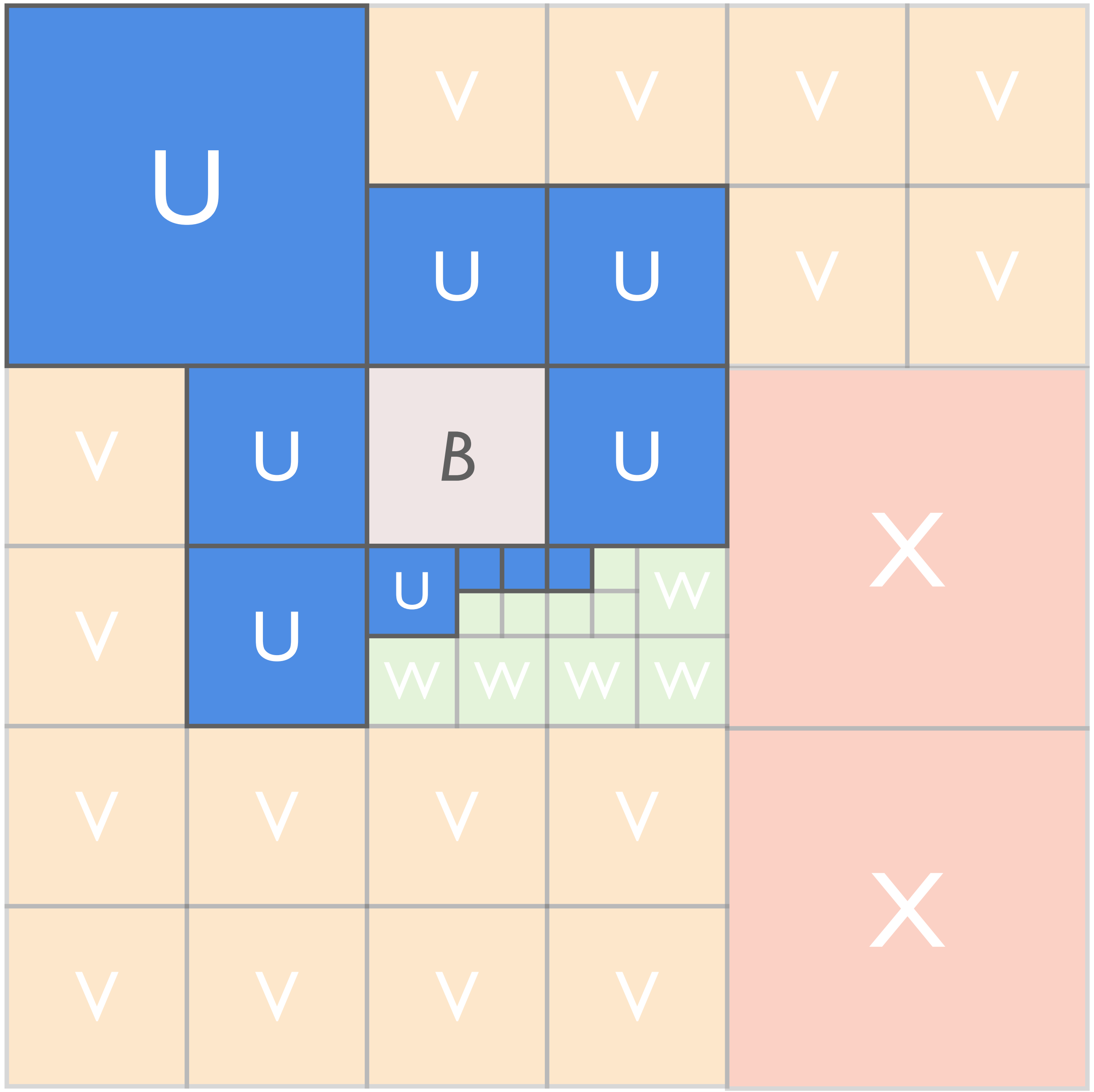


Six phases:
(1.) Upward pass
(2–5.) **List computations**
(6.) Downward pass

Phases vary in:
→ data parallelism
→ intensity (flops : mops)

Evaluation phase

Given the adaptive tree, FMM evaluation performs a series of tree traversals, doing some work at each node, *B*.



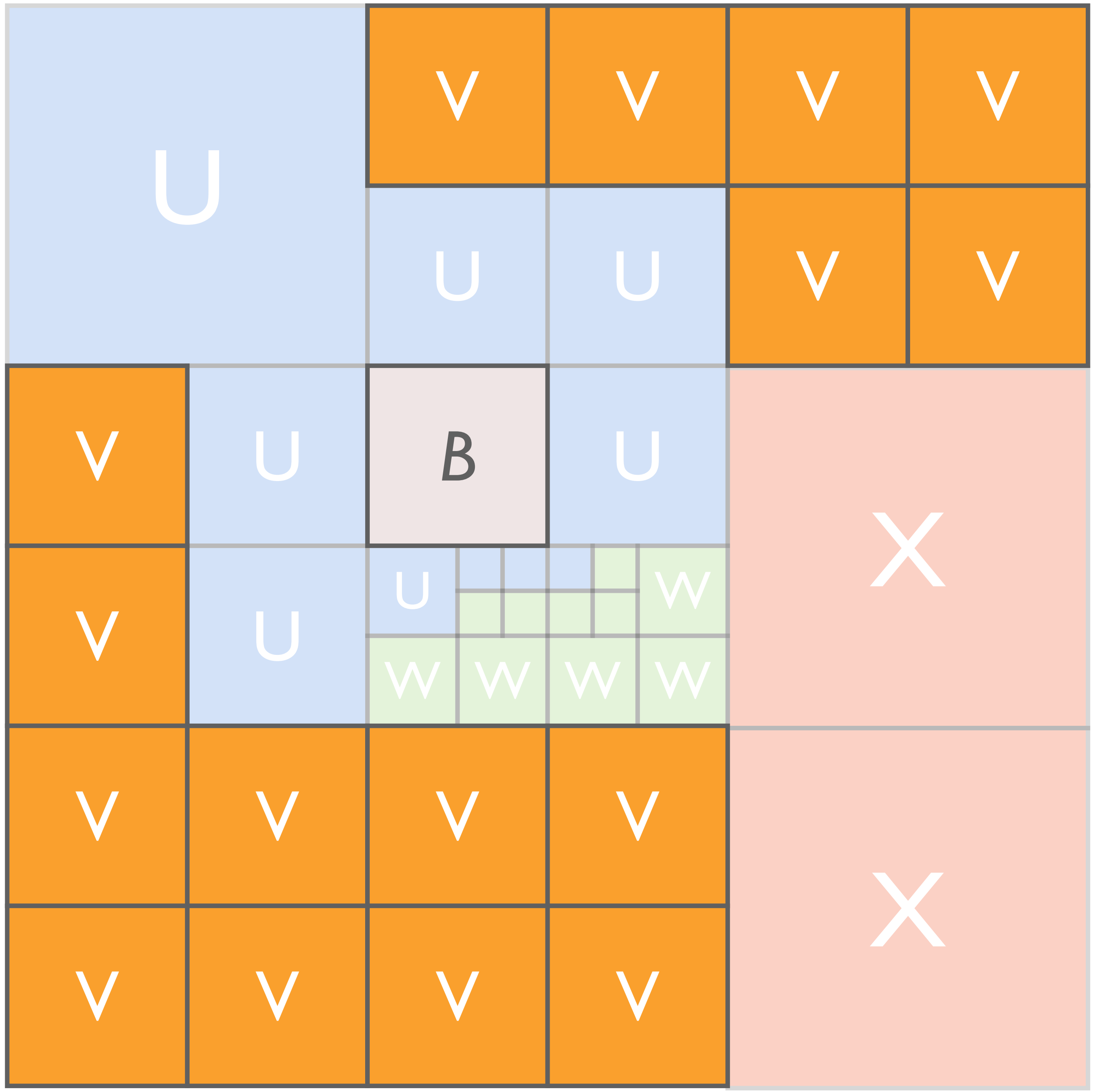
Direct $B \otimes U$:

$\rightarrow O(q^2)$ flops : $O(q)$ mops

U-List

$U_L(B\text{: } \textit{leaf})$:- neighbors (B)

$U_L(B\text{: } \textit{non-leaf})$:- empty



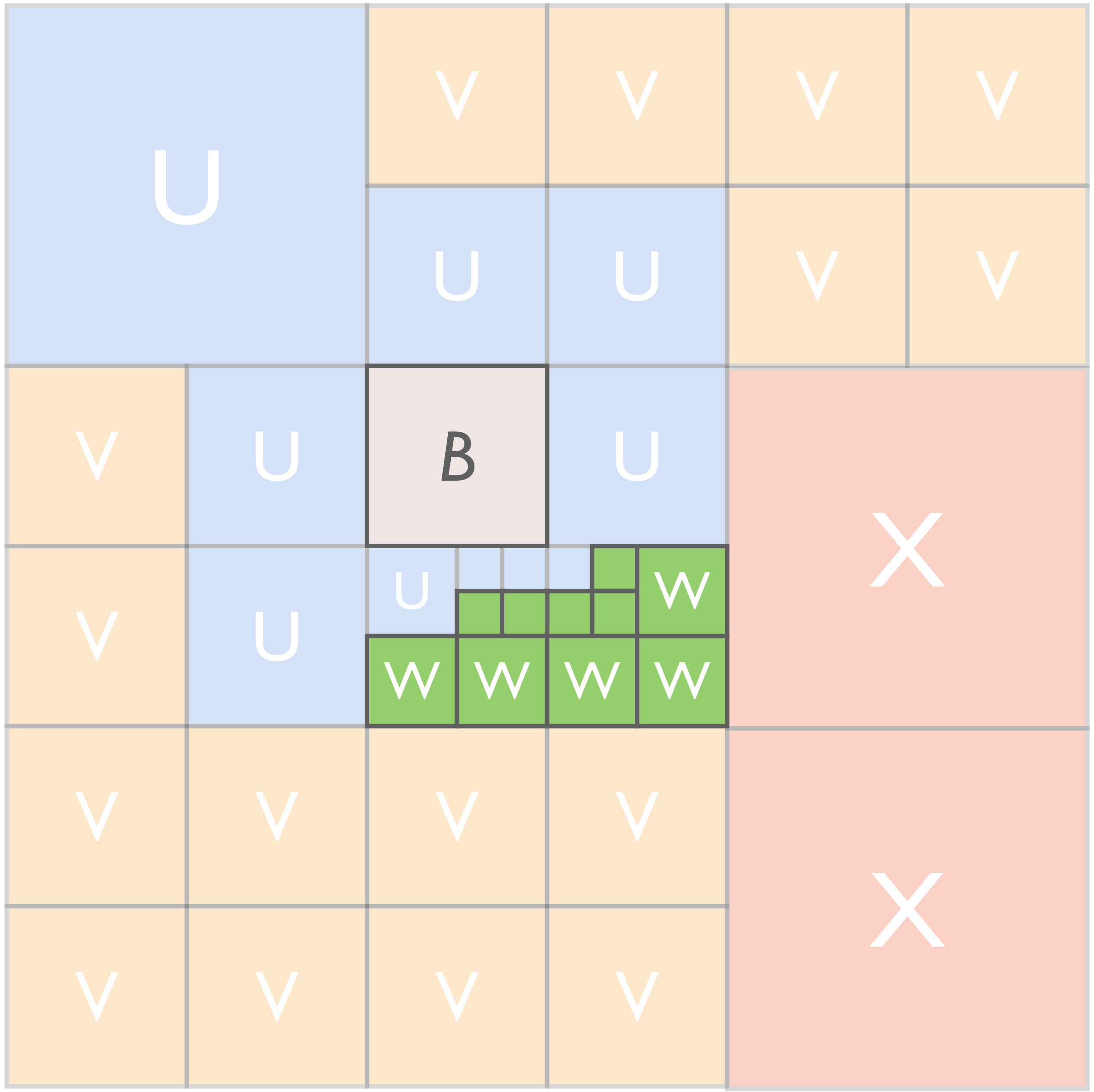
In 3D, FFTs + pointwise multiplication:

→ Easily vectorized

→ Low intensity vs. U-list

V-List

$V_L(B)$:- child (neigh (par (B))) - adj(B)

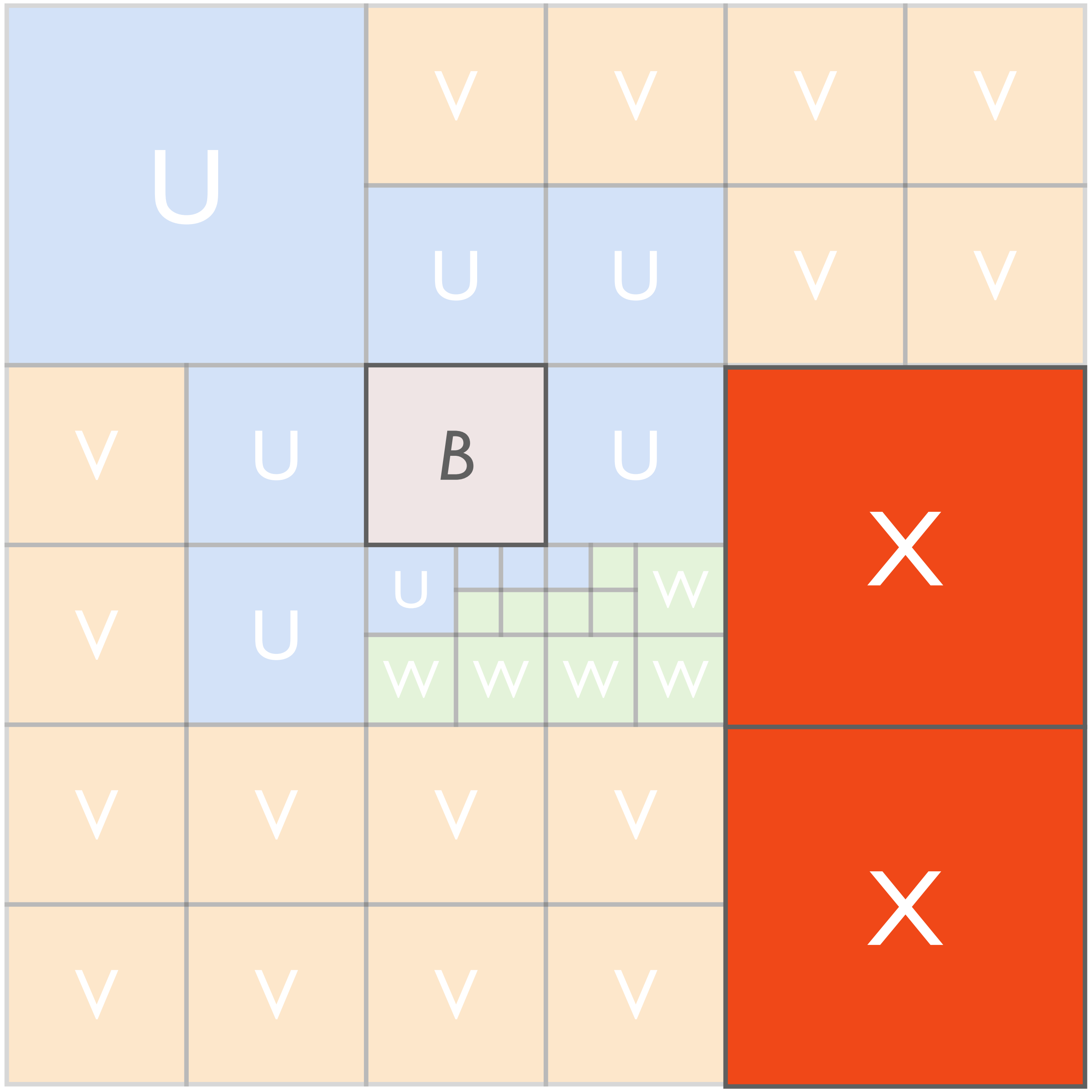


Moderate intensity

W-list

$W_L(B\text{: }leaf)$:- desc [par (neigh (B)) ∩ adj (B)] - adj (B)

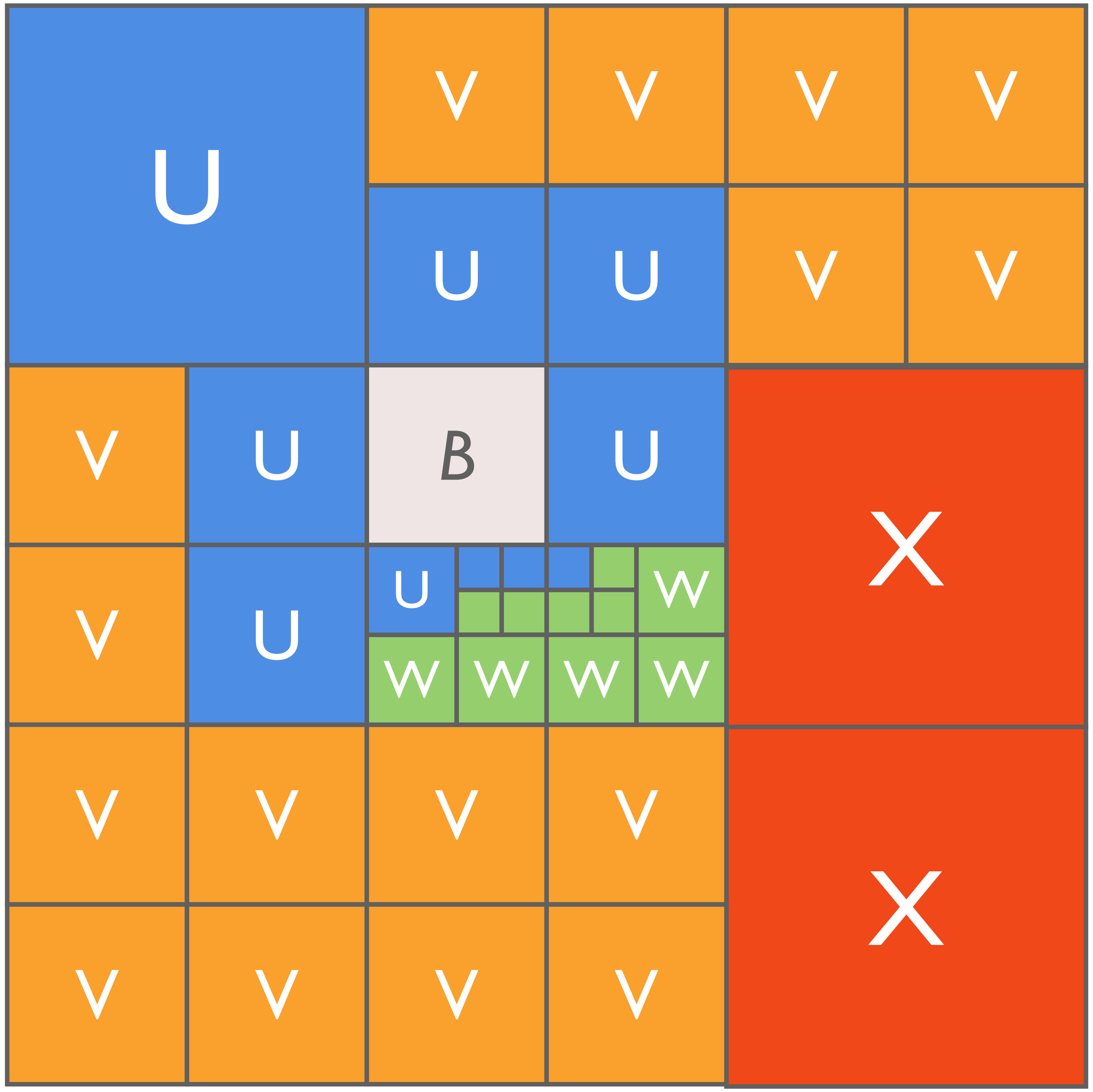
$W_L(B\text{: }non\text{-}leaf)$:- empty



Moderate intensity

X-list

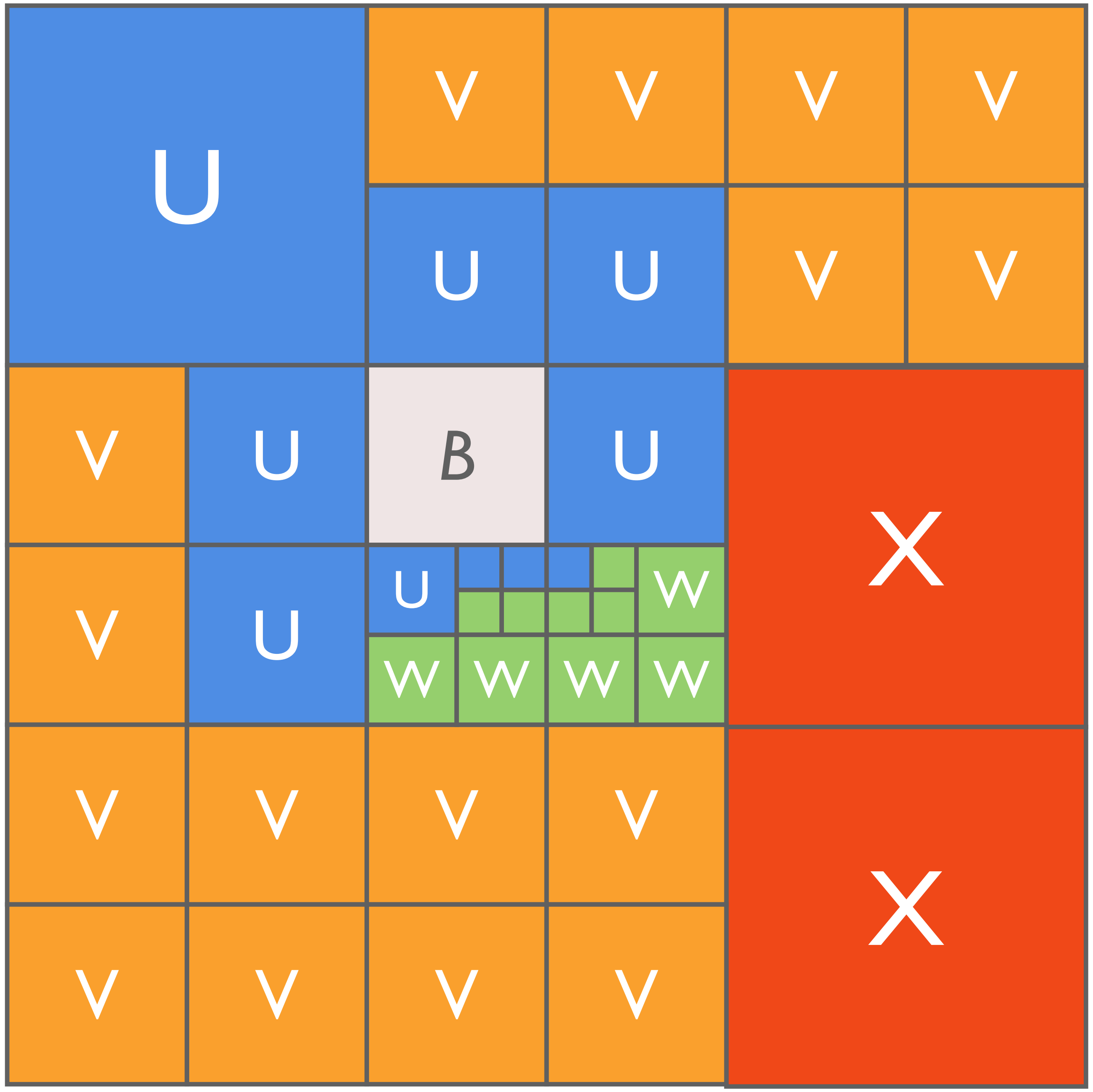
$$X_L(B) :- \{A : B \in W_L(A)\}$$



Parallelism exists:
(1) among phases, with some dependencies;
(2) within each phase;
(3) per-box.

Do not currently exploit (1).

Essence of the computation



Large q implies
→ large U-list cost, $O(q^2)$
→ cheaper V, W, X costs
(shallower tree)

Algorithmic tuning parameter, q, has a global impact on cost.

Essence of the computation

$$K(r) = \frac{C}{\sqrt{r}}$$

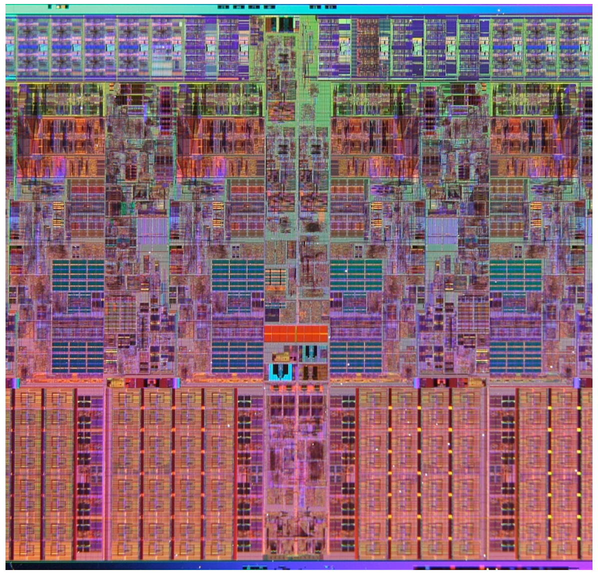
KIFMM (our variant) requires kernel evaluations with expensive flops

Essence of the computation

For instance, square-root and divide are expensive, sometimes not pipelined.

- High-performance multicore FMMs: Analysis, optimization, and tuning
  - Algorithmic characteristics
  - **Architectural implications**
  - Observations

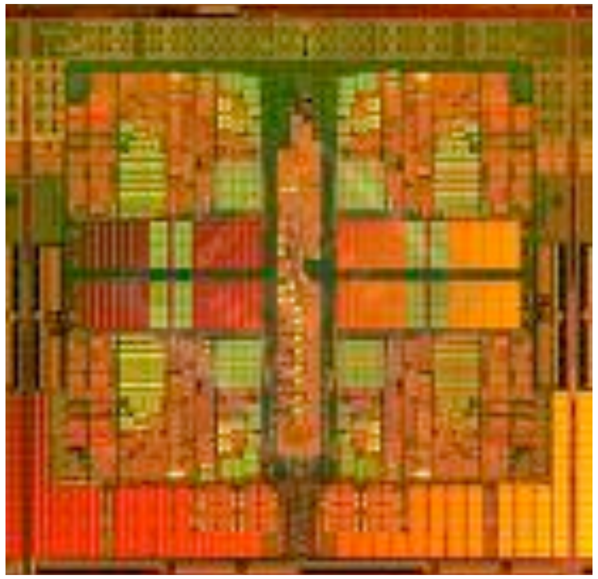
Hardware thread and core configurations



Intel X5550 "Nehalem"

2-sockets × 4-cores/socket × **2-thr/core → 16 threads**

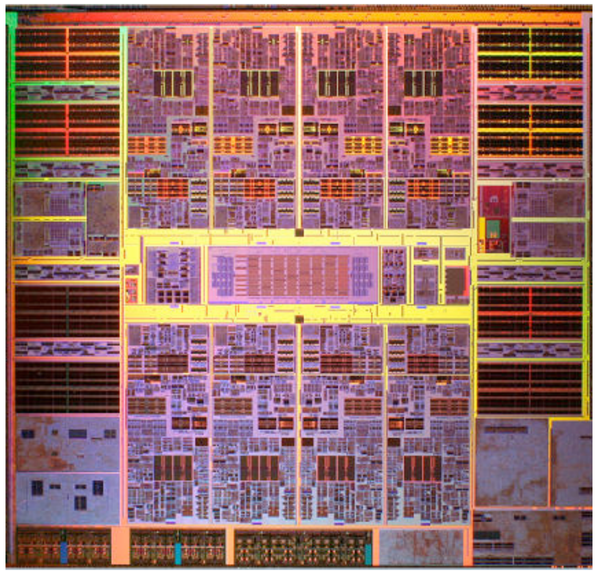
Fast **2.66 GHz** cores, **out-of-order**, **deep pipelines**.



AMD Opteron 2356 "Barcelona"

2 × 4 × **1-thr/core → 8 threads**

Fast **2.3 GHz** cores, **out-of-order**, **deep pipelines**.



Sun T5140 "Victoria Falls"

2 × 8 × **8-thr/core → 128 threads**

1.166 GHz cores, **in-order**, **shallow pipeline**.

How do they differ? What implications for FMM?

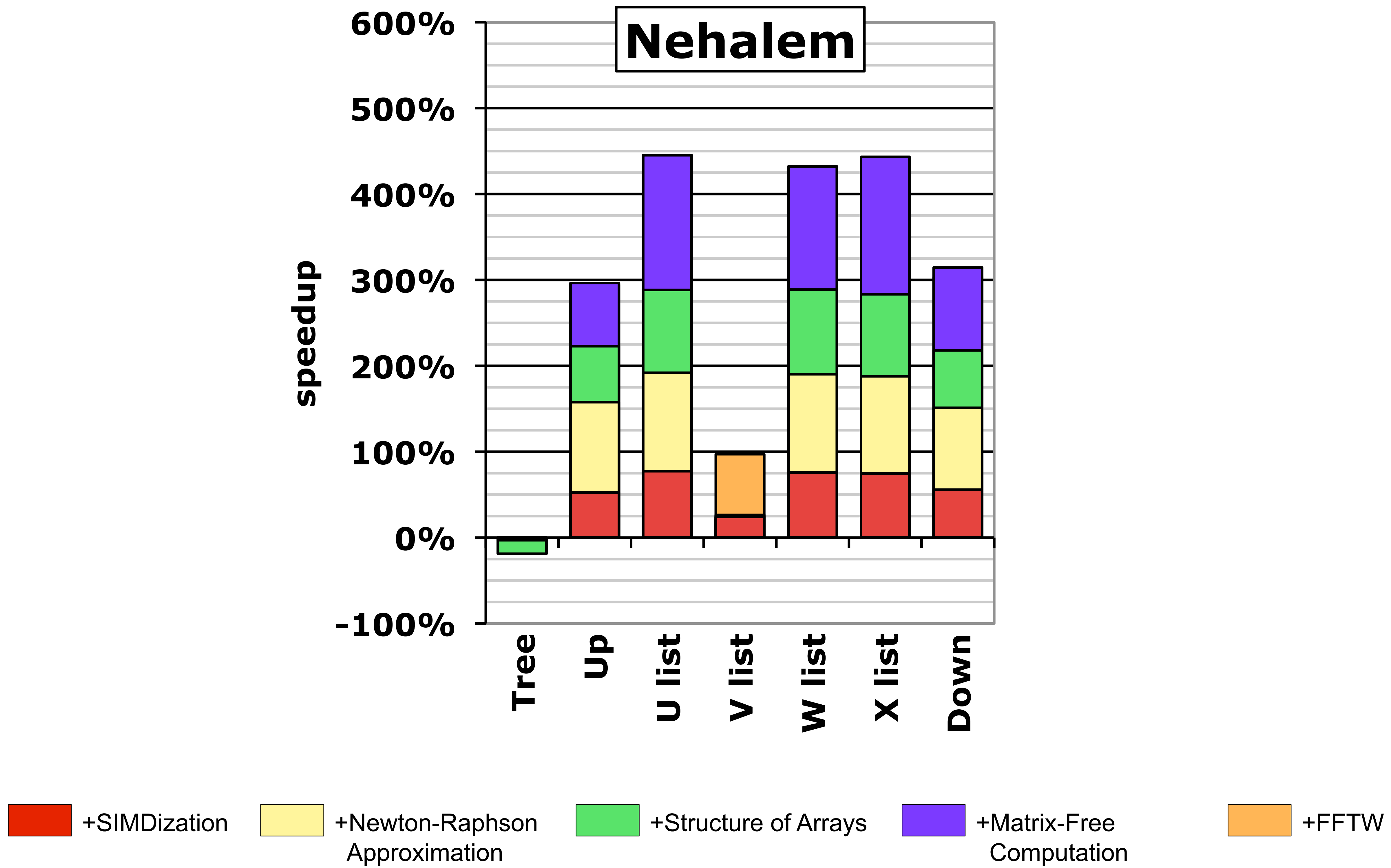
- High-performance multicore FMMs: Analysis, optimization, and tuning
  - Algorithmic characteristics
  - Architectural implications
  - **Observations**

Optimizations

- Single-core, manually coded & tuned
  - *Low-level*: SIMD vectorization (x86)
  - *Numerical*: `rsqrtps` + Newton-Raphson (x86)
  - *Data*: Structure reorg. (transpose or "SOA")
  - *Traffic*: Matrix-free via interprocedural loop fusion
  - FFTW plan optimization
- OpenMP parallelization
- Algorithmic tuning of max particles per box, q

Single-core Optimizations

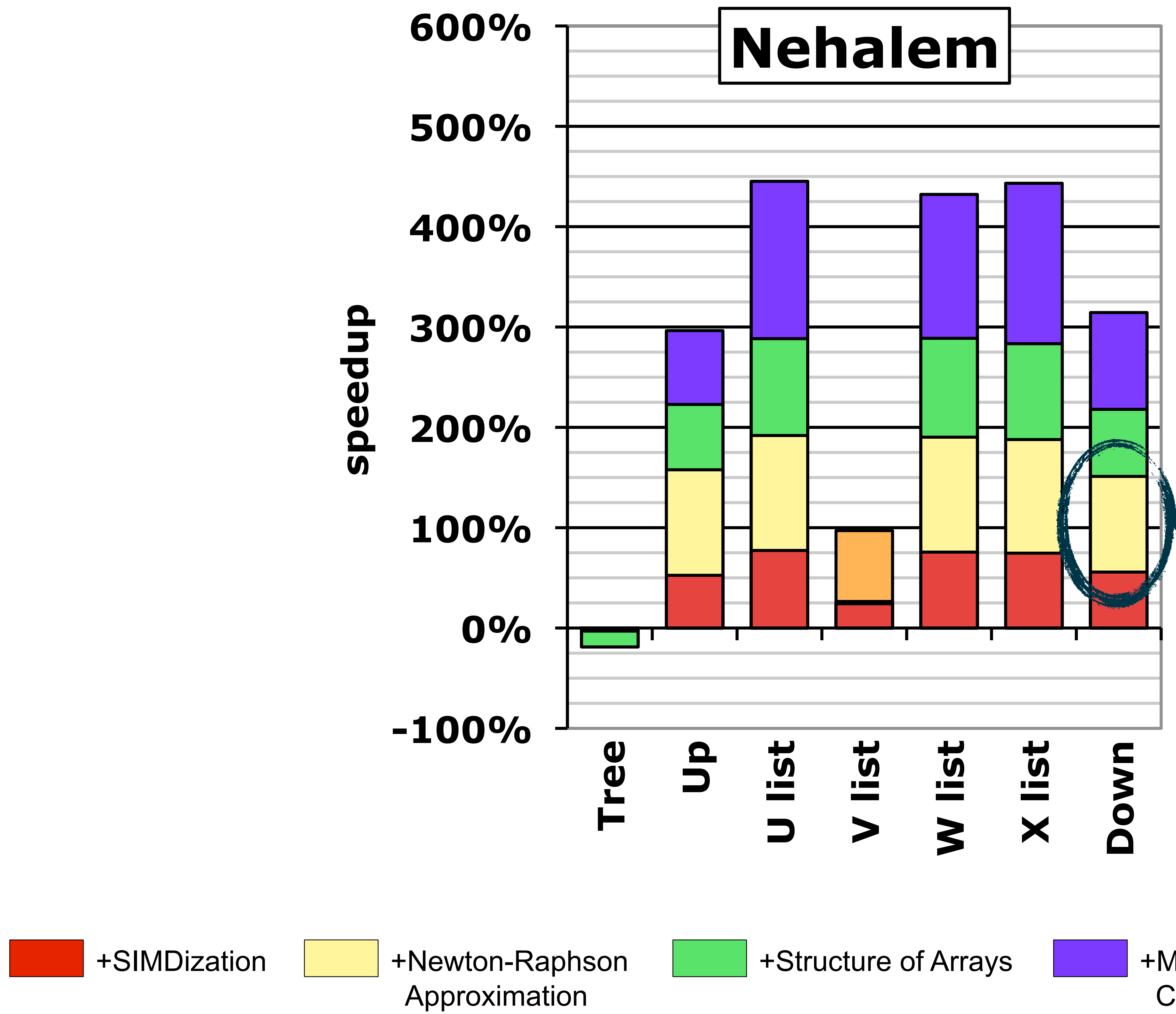
$N_s = N_t$ = 4M, Double-Precision, Non-uniform (ellipsoidal)



Reference: kifmm3d [Ying, Langston, Zorin, Biros]

Single-core Optimizations

$N_s = N_t =$ 4M, Double-Precision, Non-uniform (ellipsoidal)



SIMD → 85.5 (double), **170.6** (single) **Gflop/s**

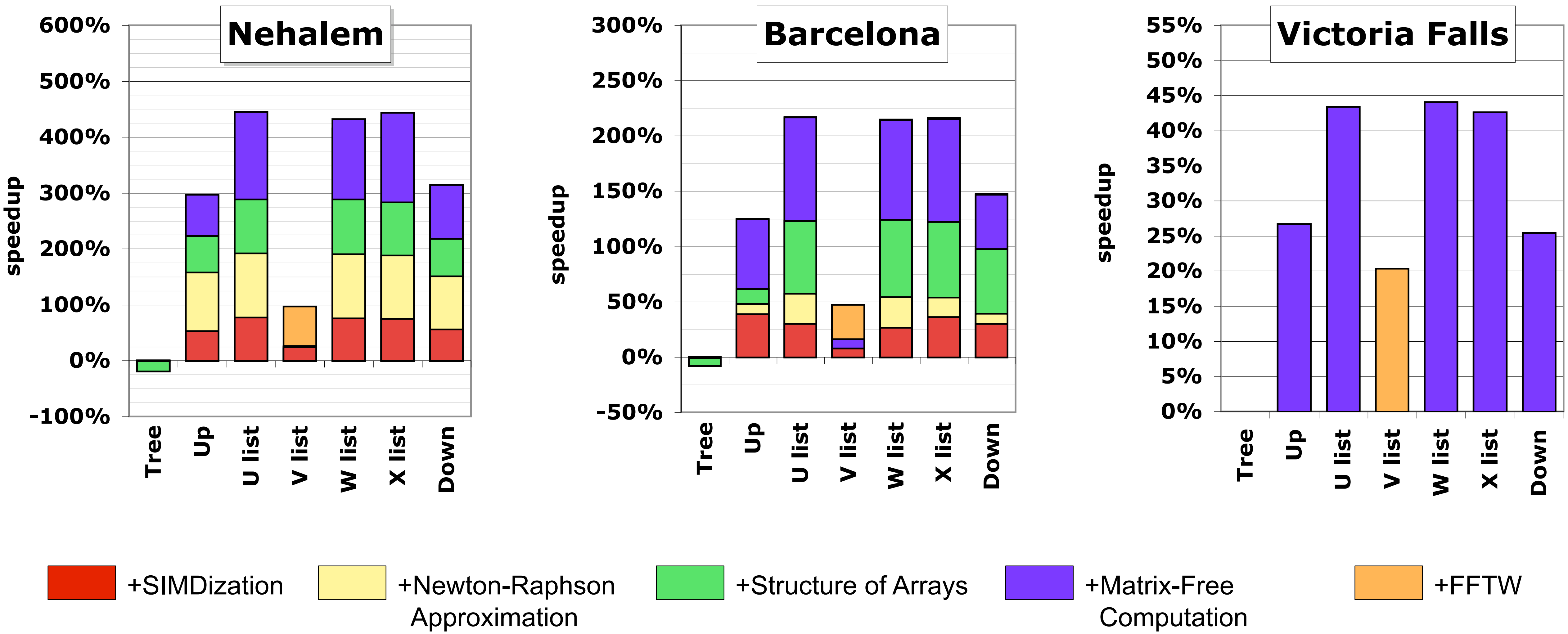
Reciprocal square-root → 0.853 (double), **42.66** (single) **Gflop/s**

x86 has fast approximate *single-precision* rsqrt, *exploitable in double.*

Single-core Optimizations

$N_s = N_t$ = 4M, Double-Precision, Non-uniform (ellipsoidal)

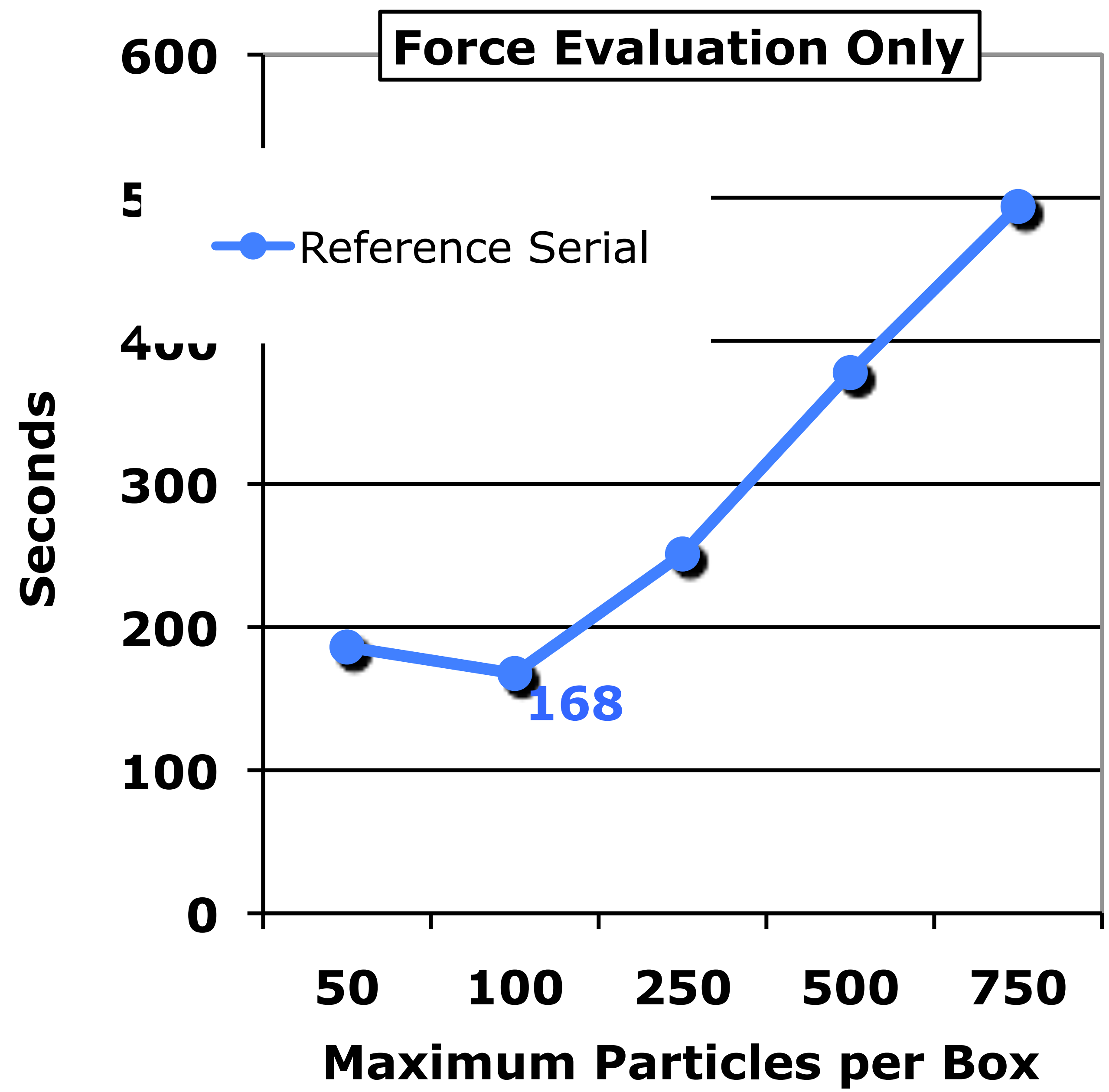
~ 4.5x *~ 2.2x* *~ 1.4x*



Less impact on Barcelona (why?) and Victoria Falls.

Algorithmic Tuning of *q* = Max pts / box

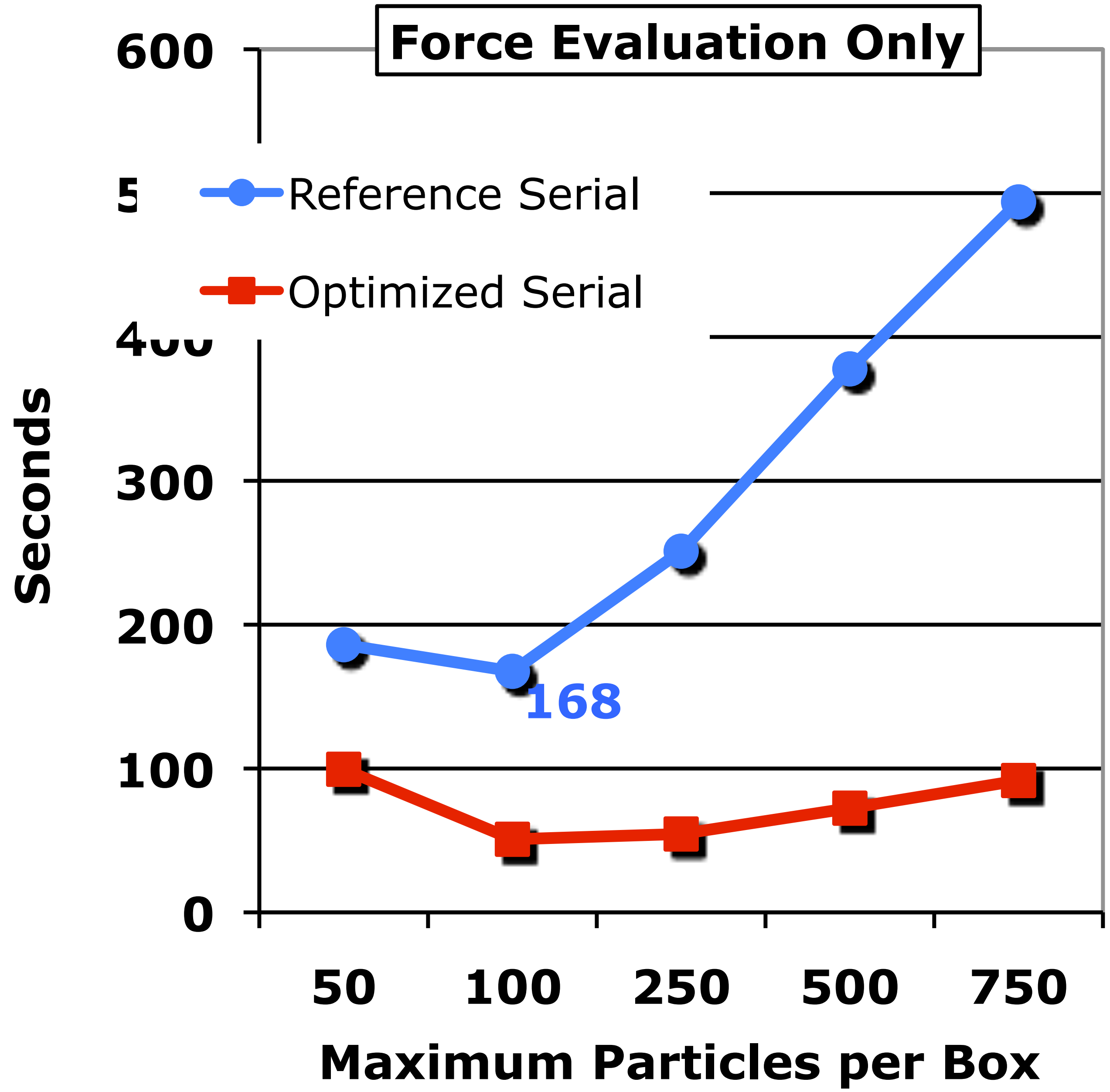
Nehalem



Tree shape and relative component costs vary as q varies.

Algorithmic Tuning of *q* = Max pts / box

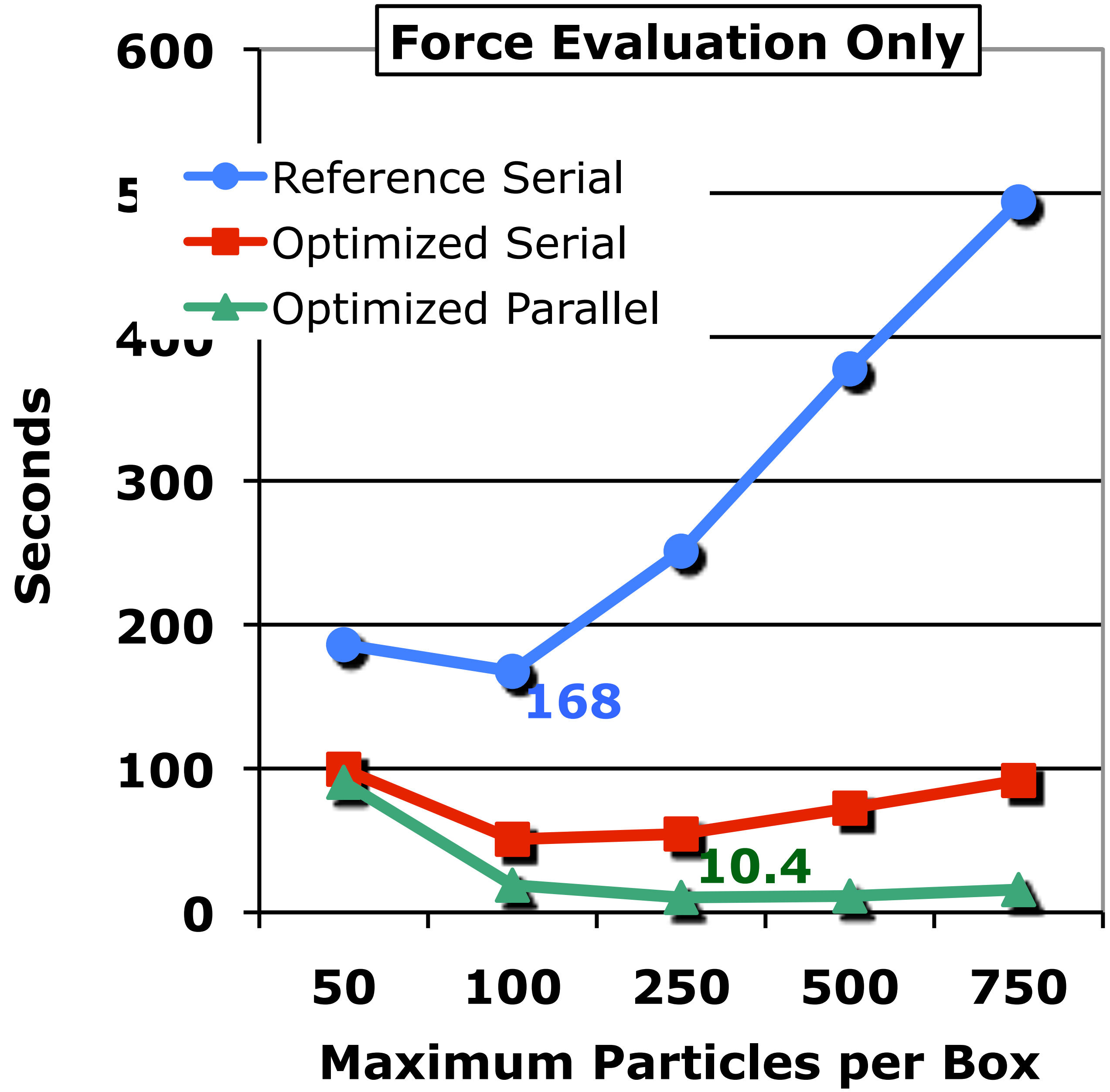
Nehalem



Shape of curve changes as we introduce optimizations.

Algorithmic Tuning of *q* = Max pts / box

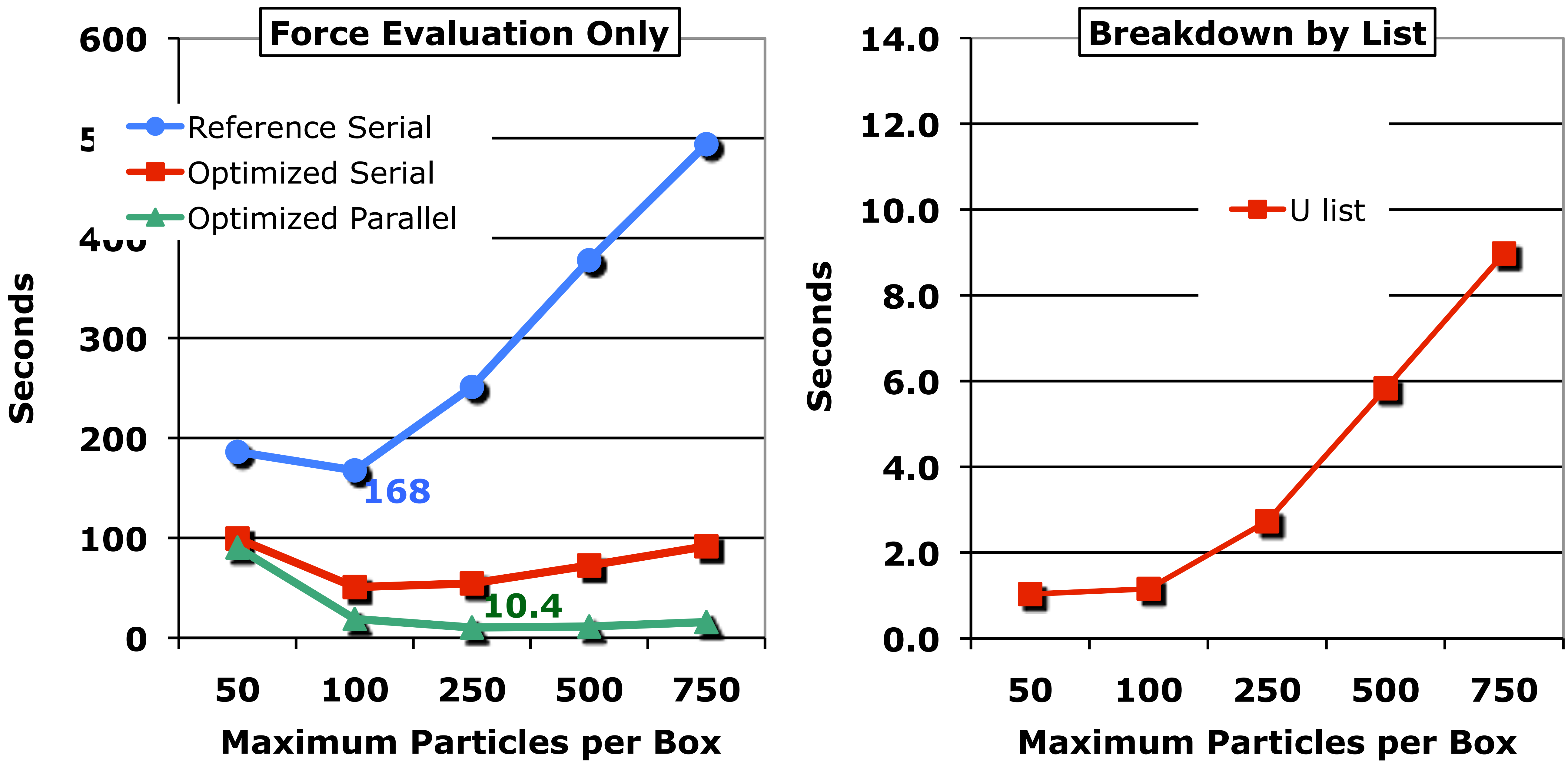
Nehalem



Shape of curve changes as we introduce optimizations.

Algorithmic Tuning of q = Max pts / box

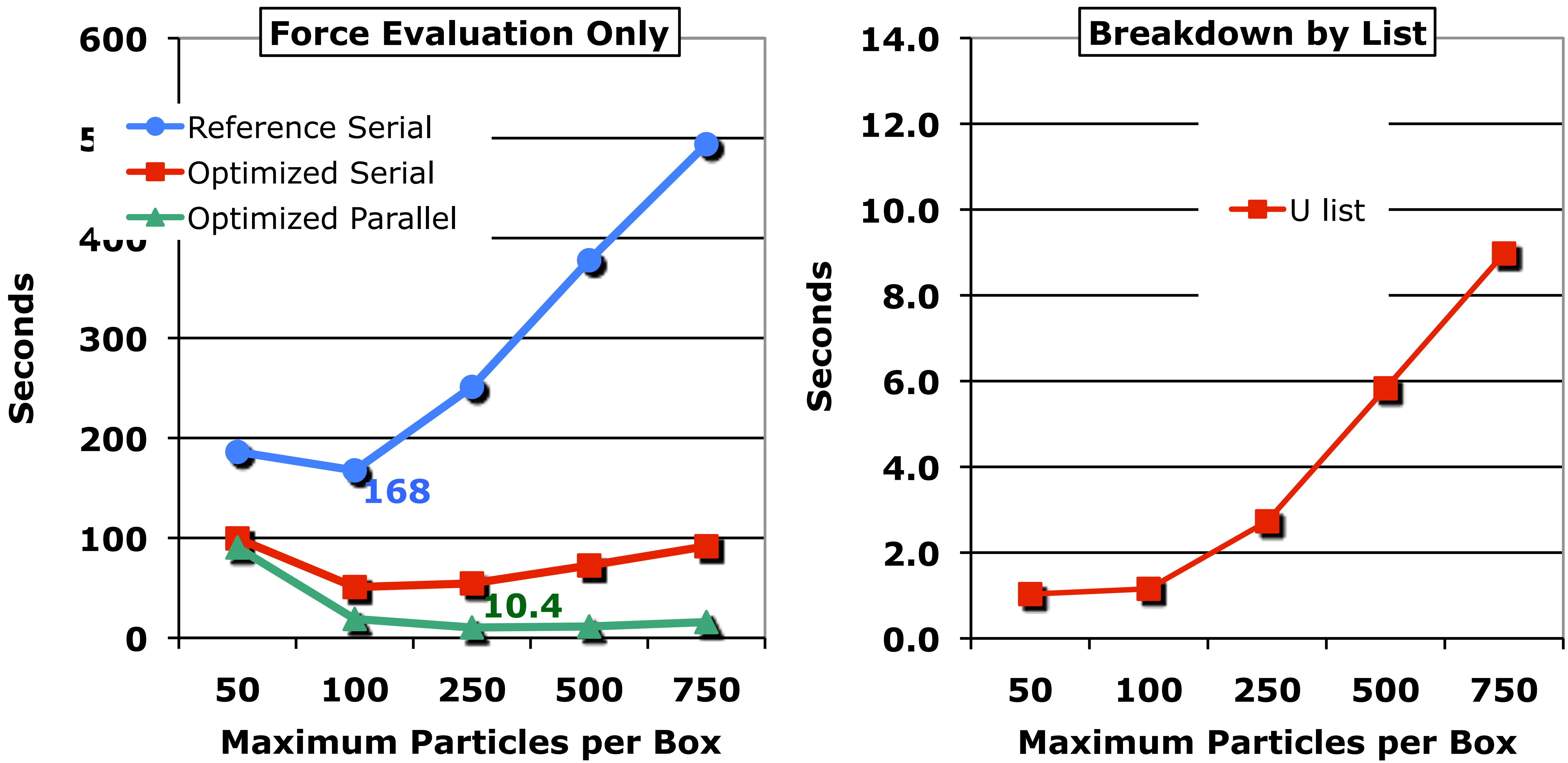
Nehalem



Why? Consider phase costs for the "Optimized Parallel" implementation.

Algorithmic Tuning of *q* = Max pts / box

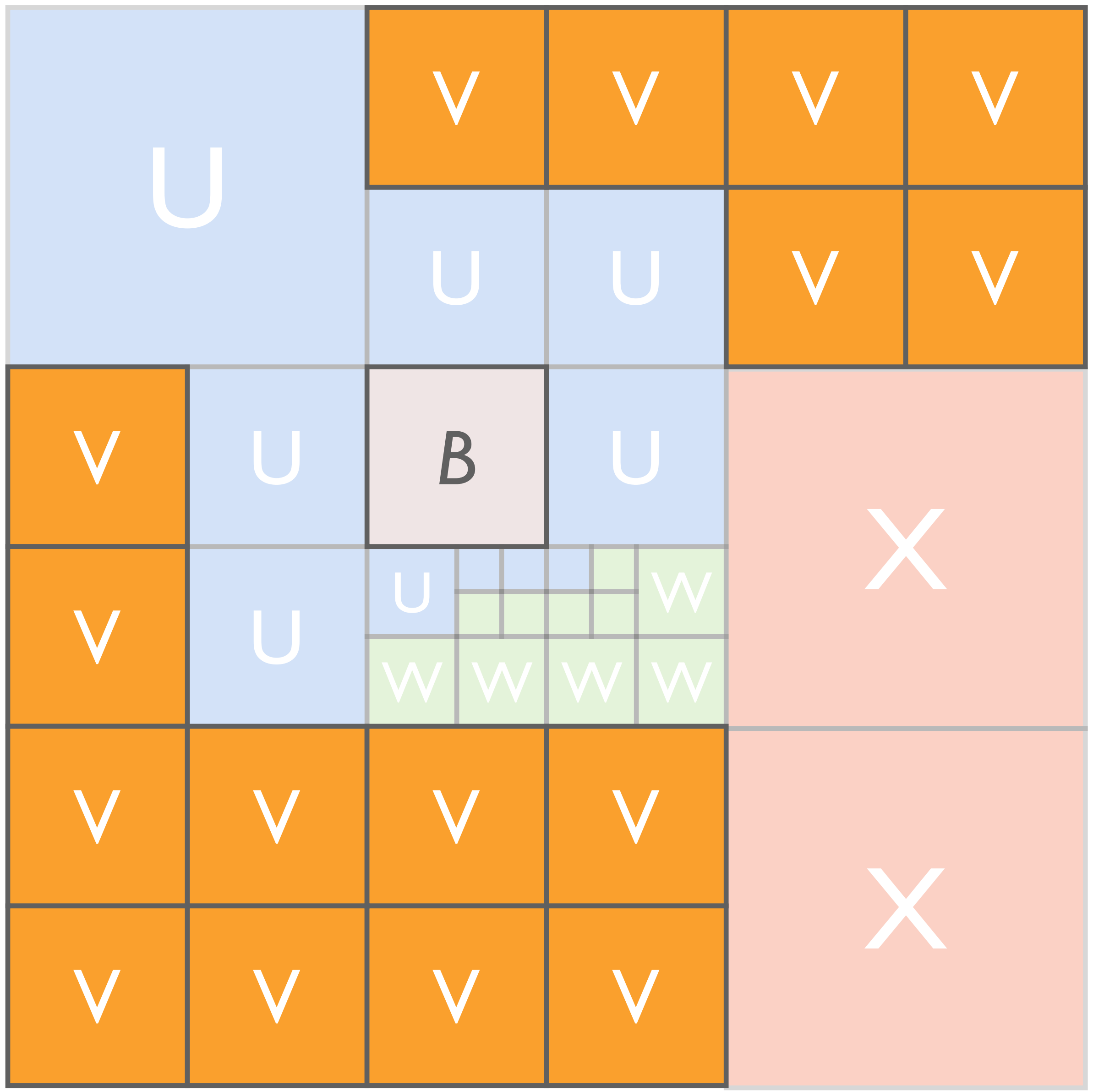
Nehalem

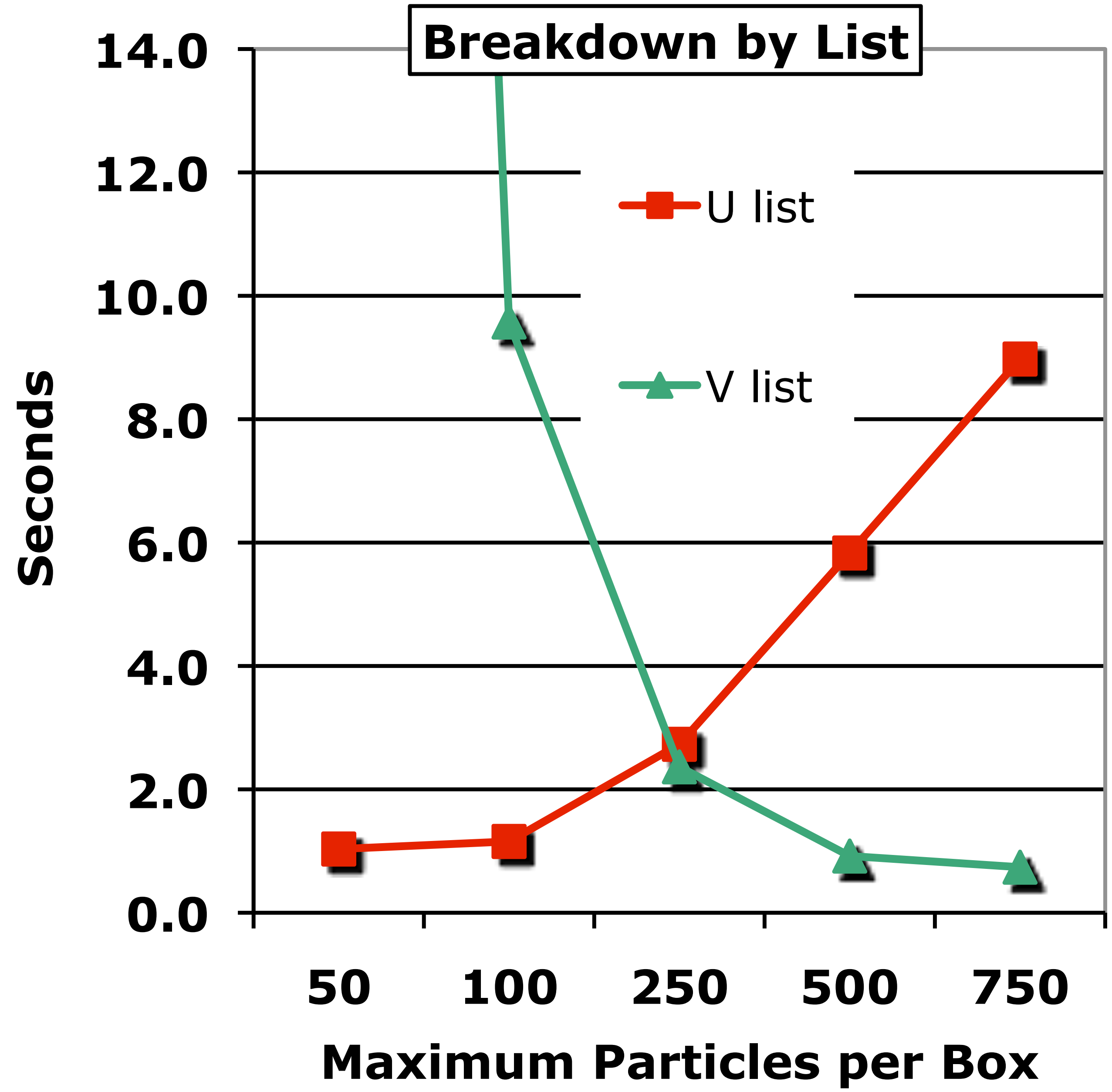


Recall: Cost(U-list) ~ $O(q^2)$ per box

Algorithmic Tuning of *q* = Max pts / box

Nehalem

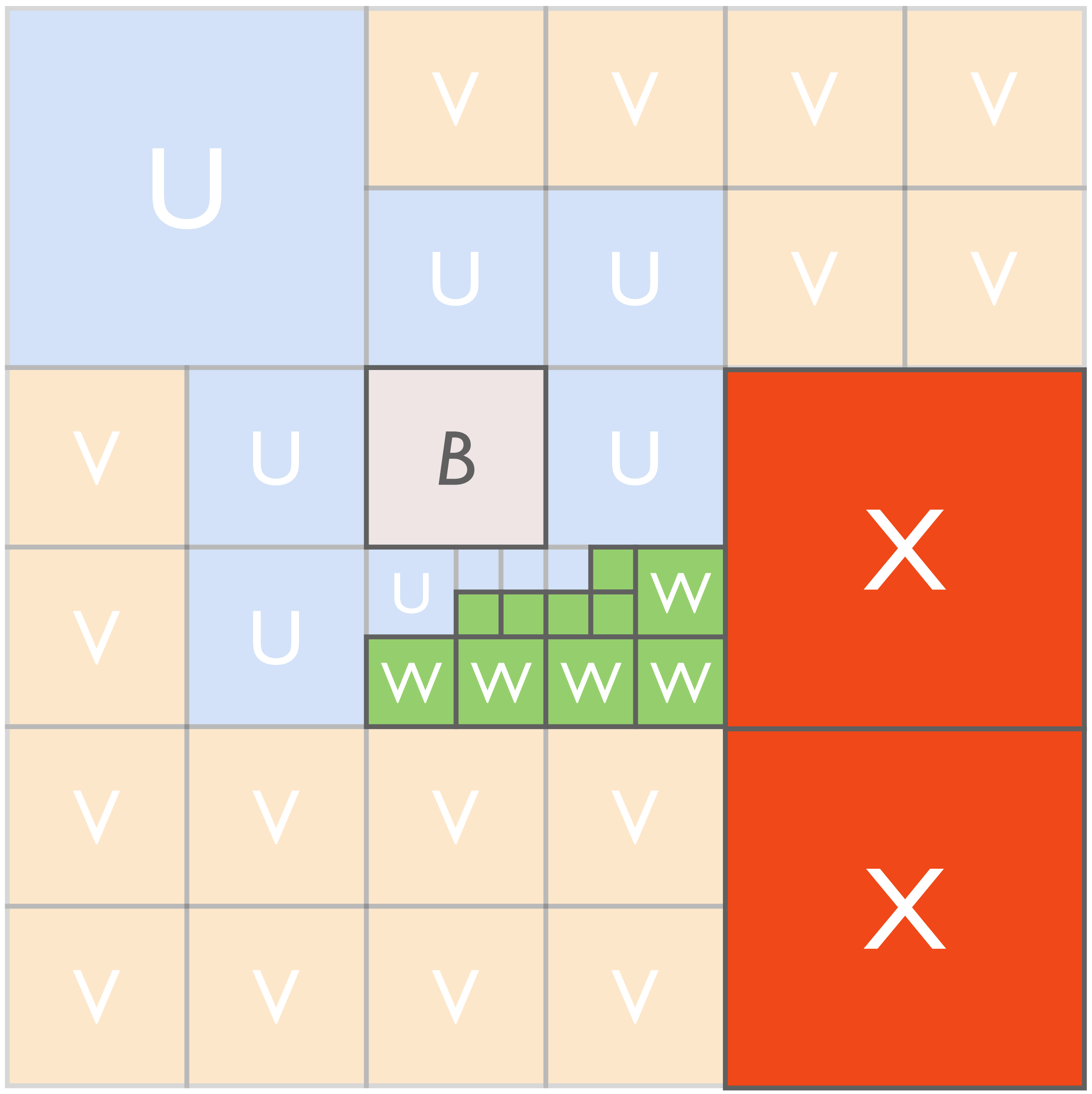


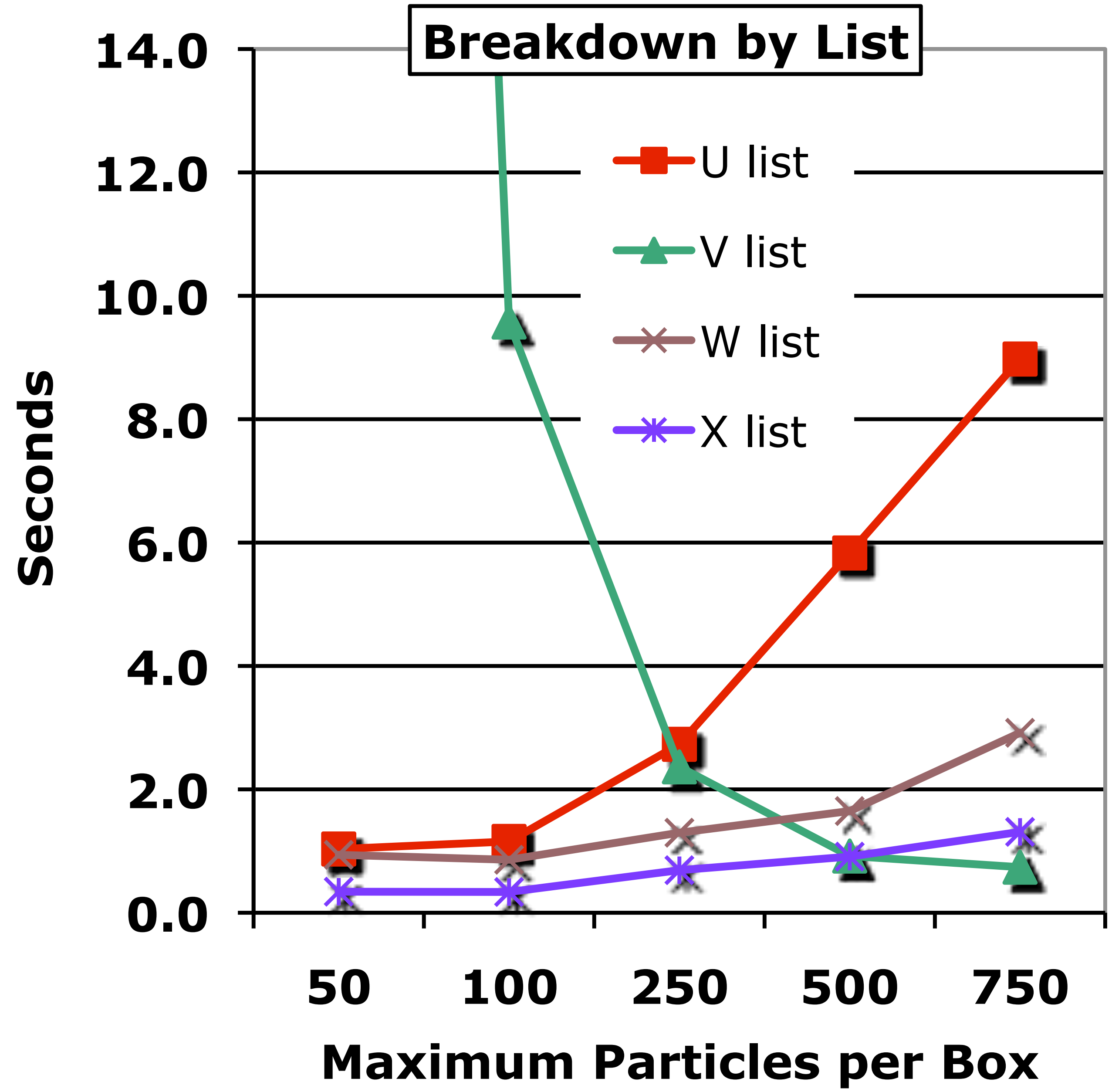


A more shallow tree reduces cost of V-list phase.

Algorithmic Tuning of q = Max pts / box

Nehalem

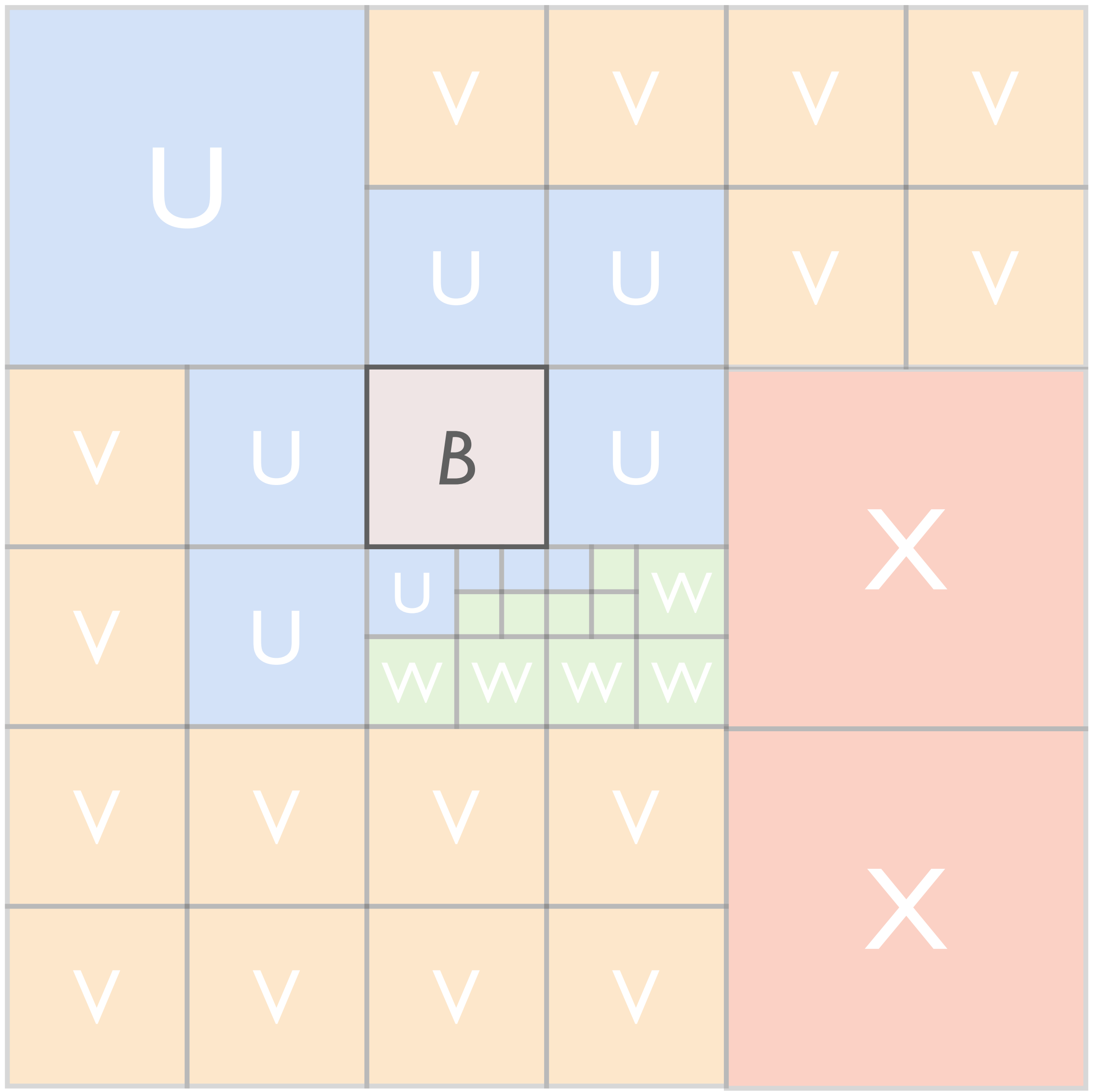


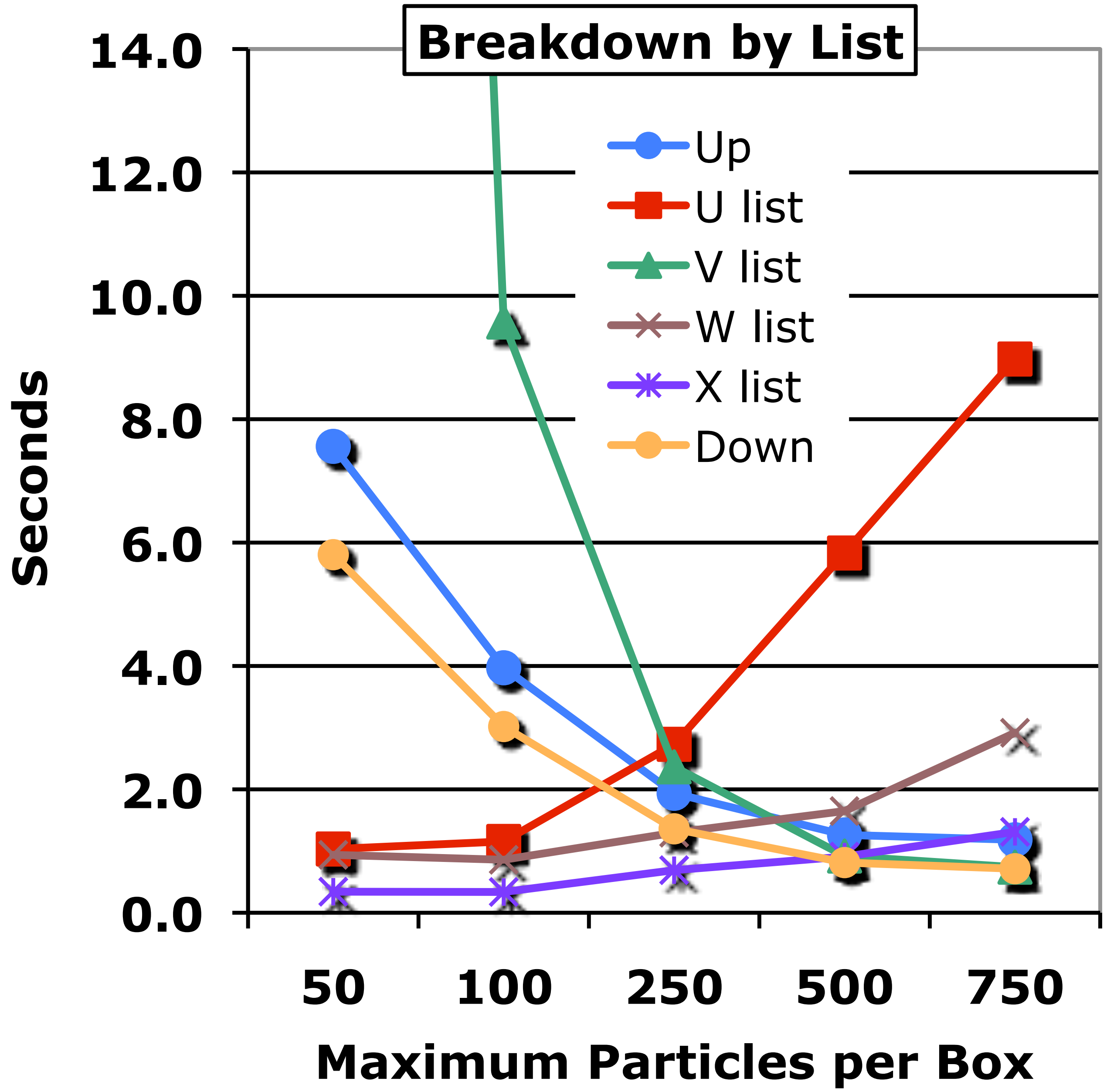


Computational intensity of W, X more like U than V.

Algorithmic Tuning of *q* = Max pts / box

Nehalem

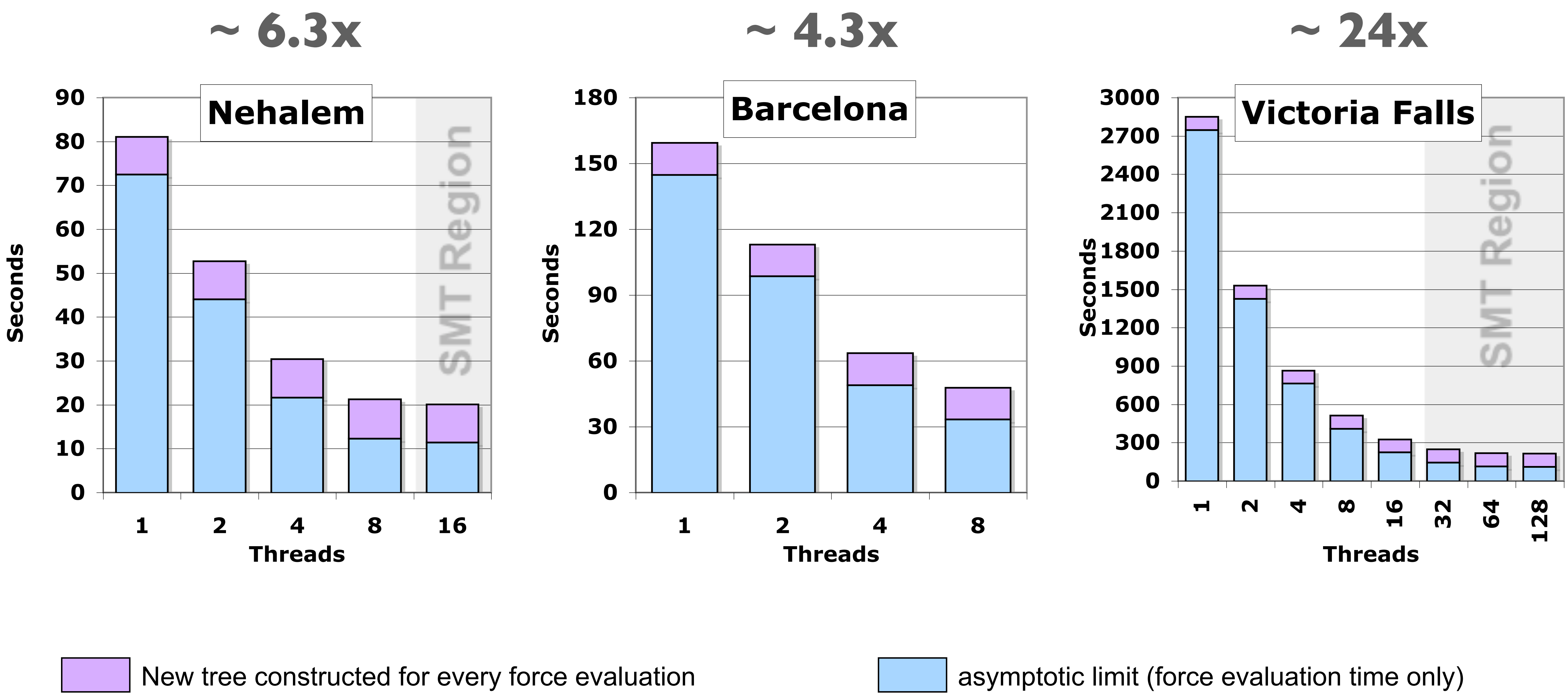




Optimal q will vary as the point distribution varies.

Multicore Scalability over Optimized Baseline

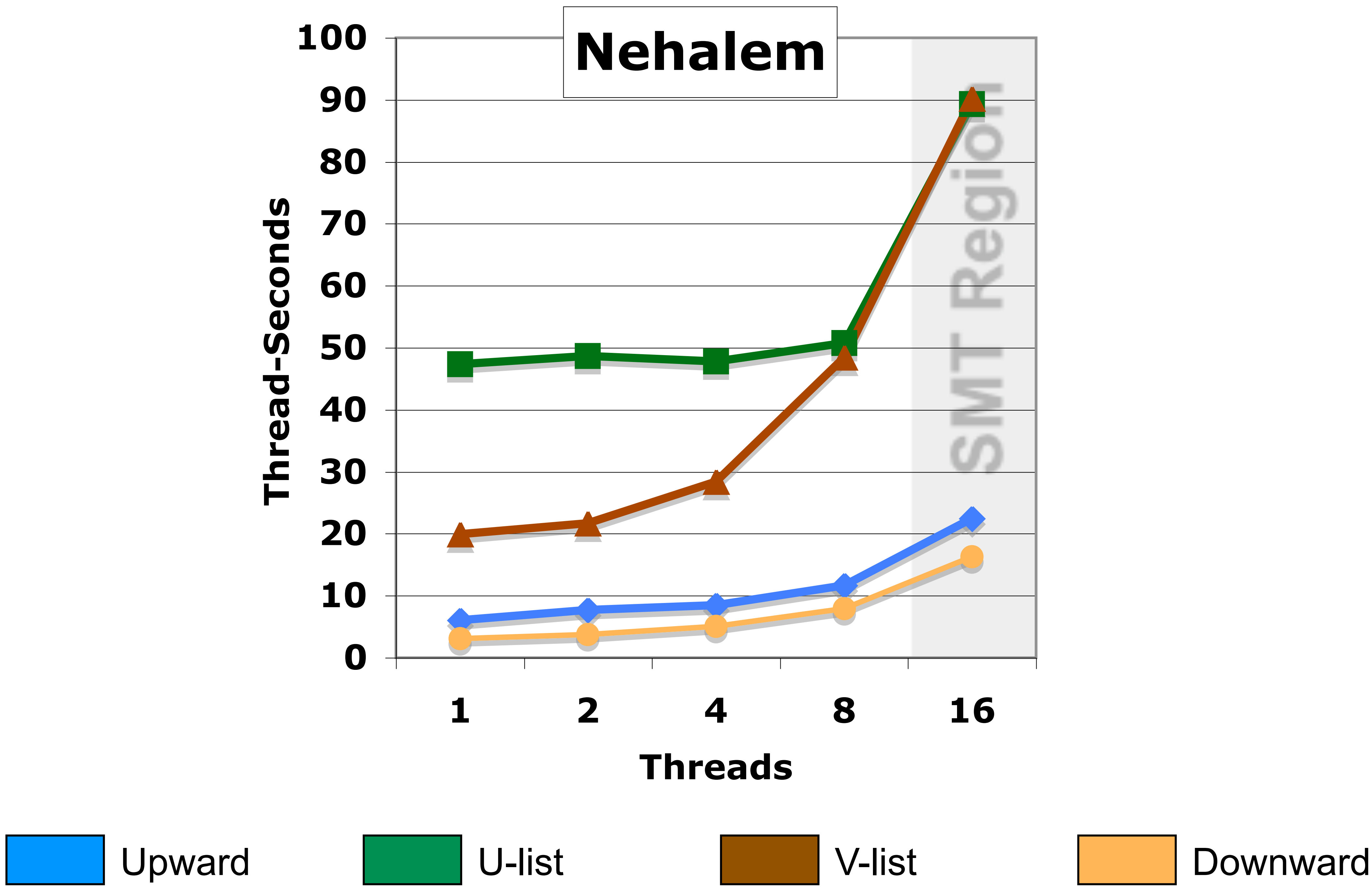
Ellipsoidal Distribution



Need to improve tree construction. Little benefit from SMT.

Efficiency, via Parallel Cost – $p \cdot T_p$

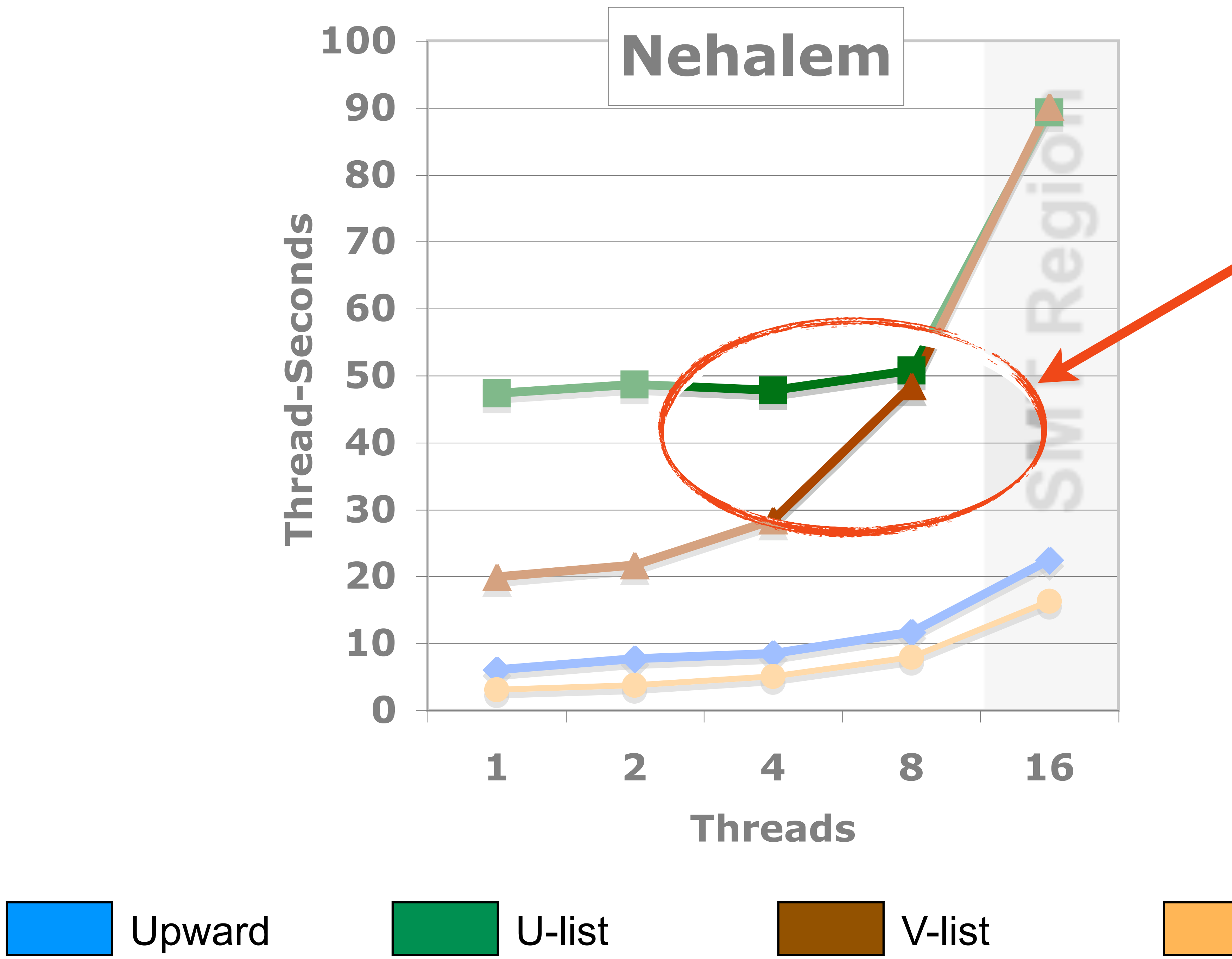
Uniform Distribution



Flat horizontal line = perfect scaling

Efficiency, via Parallel Cost – $p \cdot T_p$

Uniform Distribution



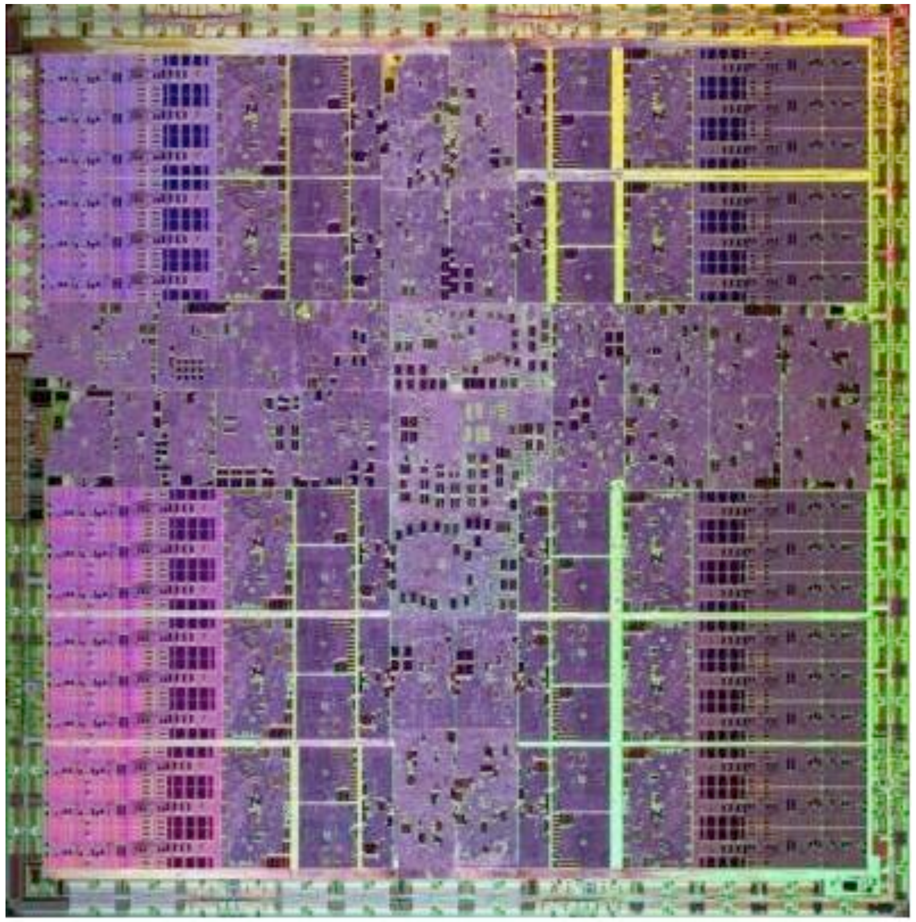
Hypothesis: Contention.

Idea: Could overlap U & V lists

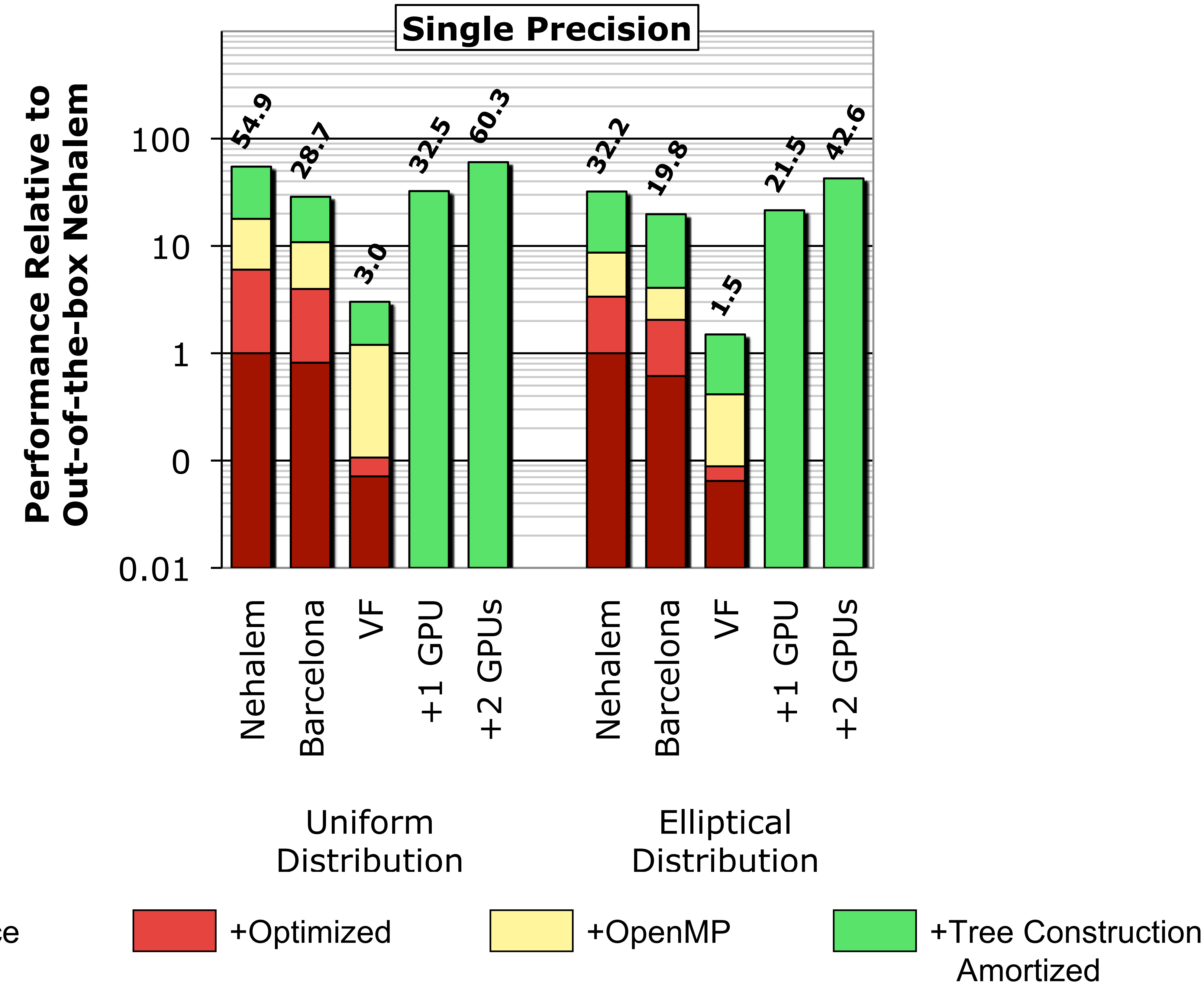
Flat horizontal line = perfect scaling

GPU comparison: NVIDIA T10P

- Our prior work on MPI+CUDA Lashuk, et al., SC'09
- System: NCSA Lincoln Cluster
  - Dual-socket Xeon
  - 1 node, 1 MPI task per socket & GPU (tasks mostly idle)
  - 1- and 2-GPU configs
  - Single-precision only for now
- **12x compute + 5x bandwidth**

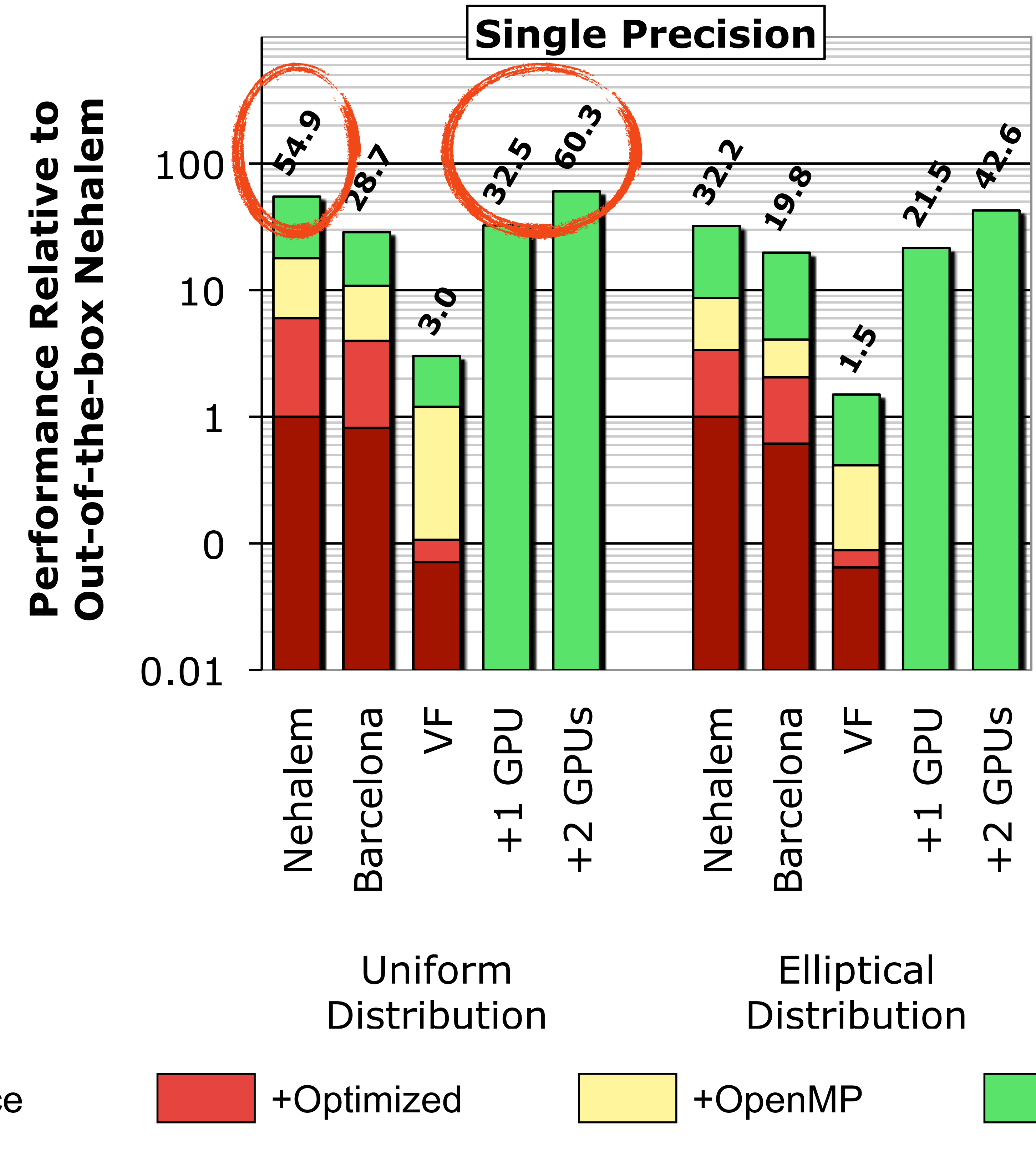


Cross-Platform Performance Comparison (Summary)



Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.

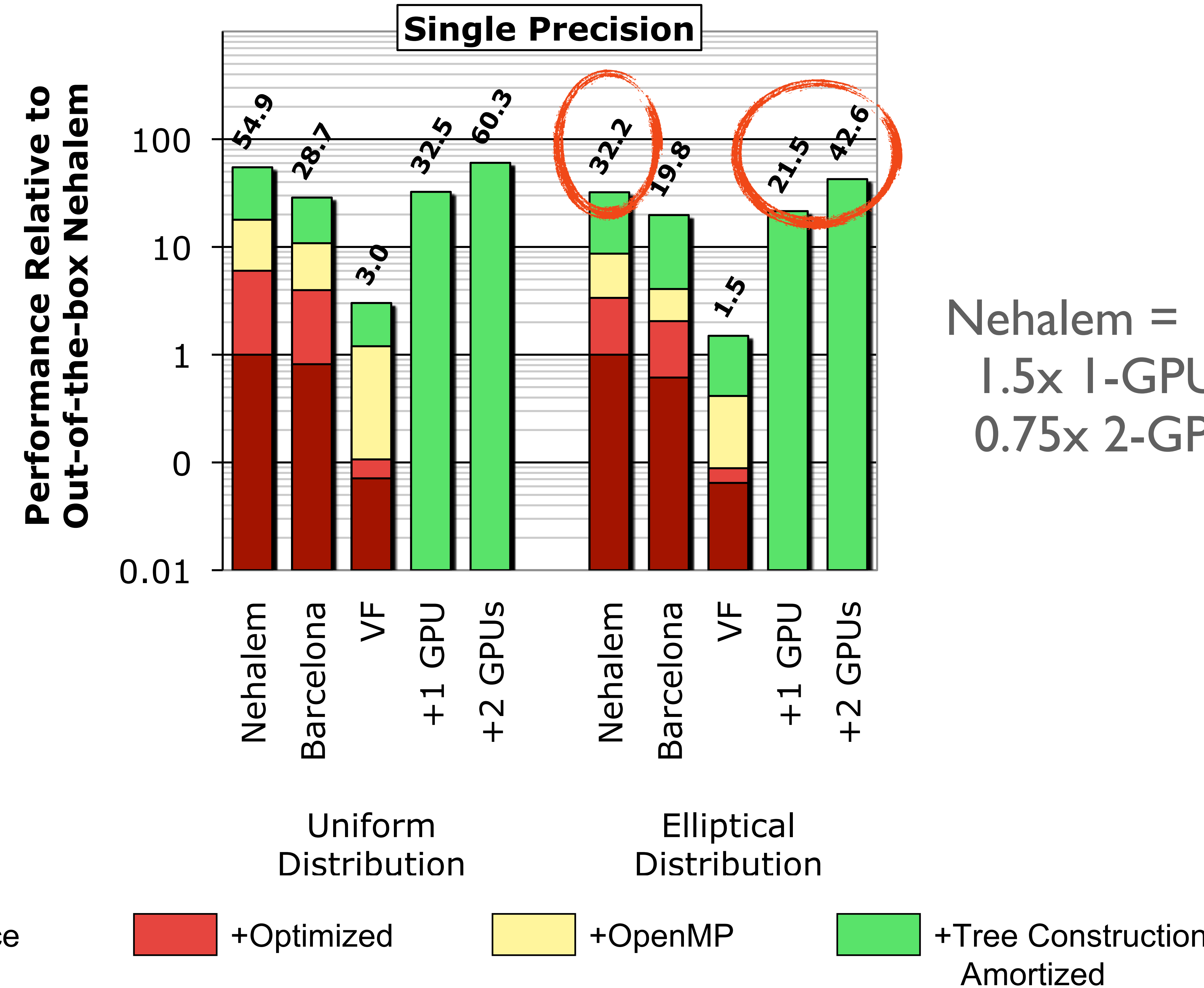
Cross-Platform Performance Comparison (Summary)



Nehalem =
1.7x 1-GPU
0.9x 2-GPU

Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.

Cross-Platform Performance Comparison (Summary)

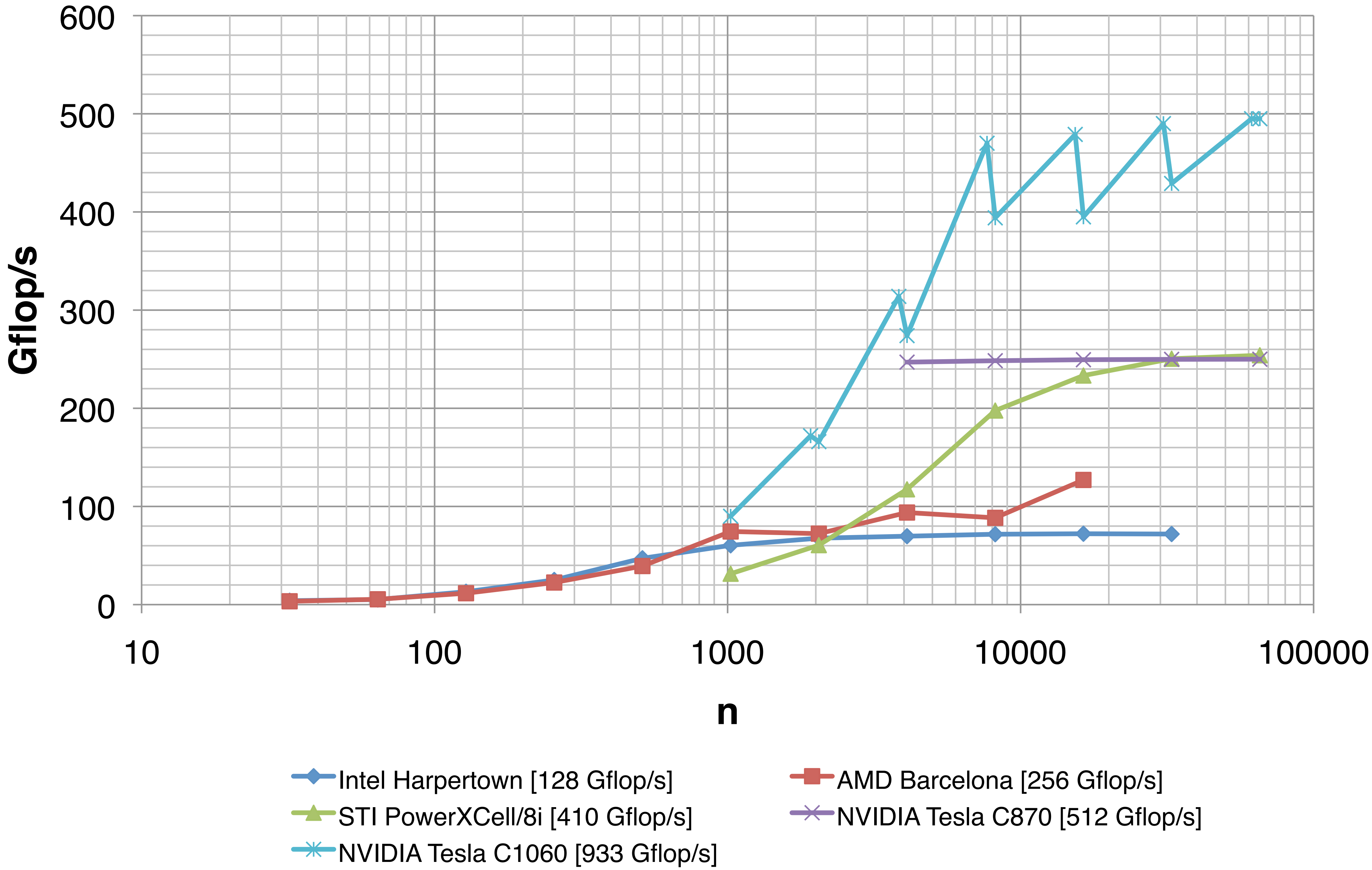


Nehalem =
1.5x 1-GPU
0.75x 2-GPU

Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.

Performance of Direct n-body Computation

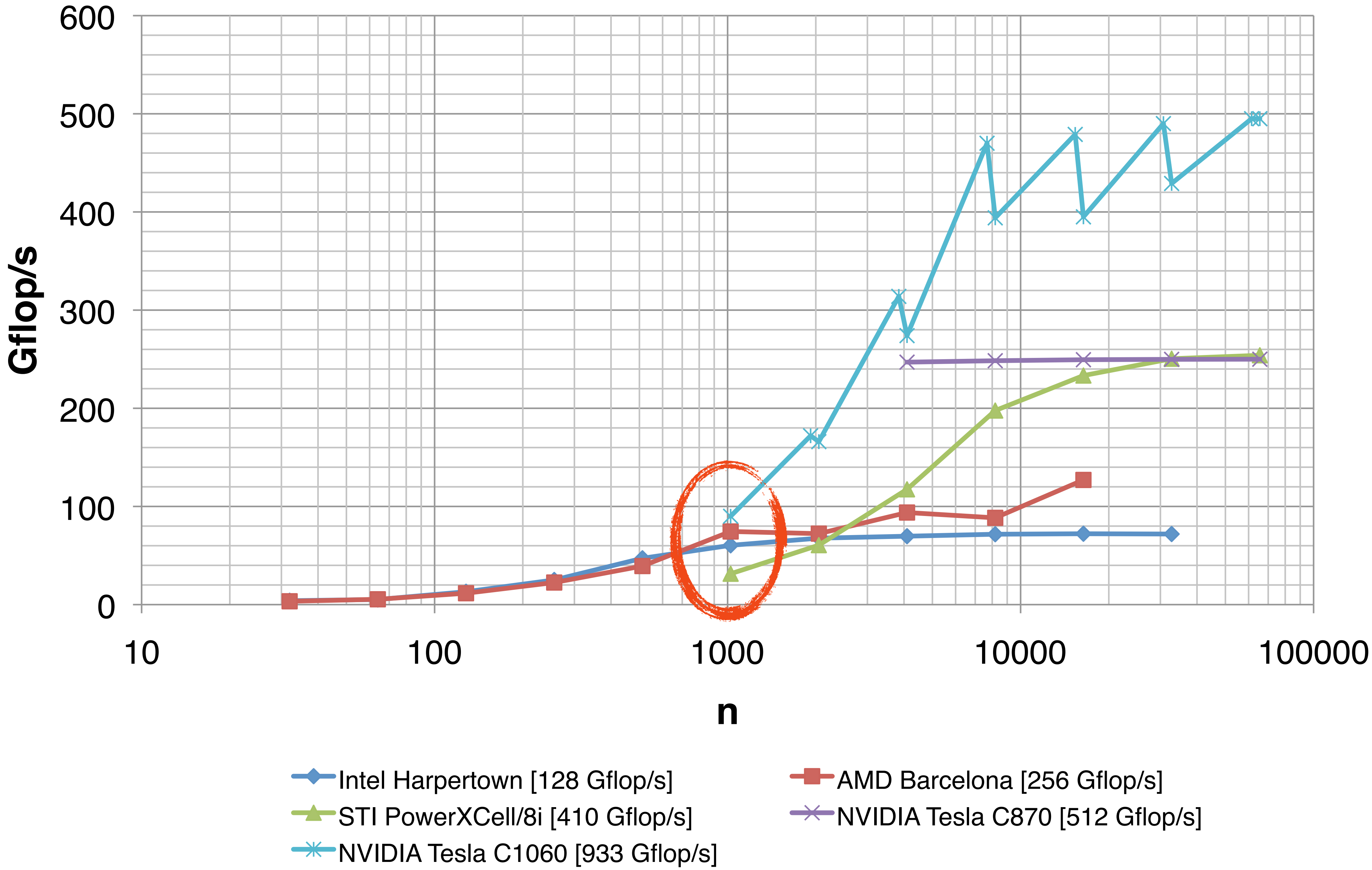
Single Precision



GPU achieves ~50% of the theoretical peak for large n.

Performance of Direct n-body Computation

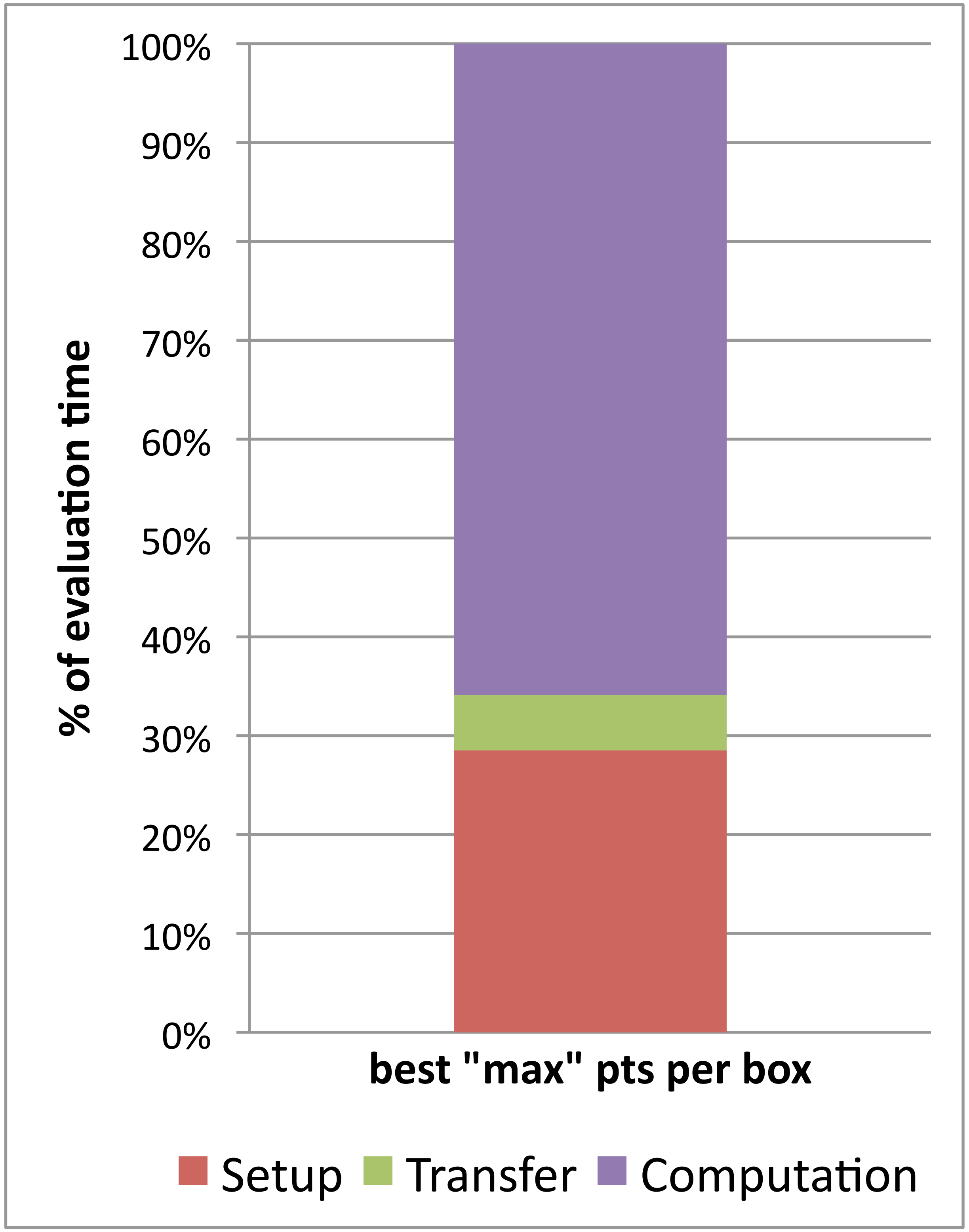
Single Precision



Competing implementations have comparable performance for small n (optimal for FMM).

Decomposition of GPU time

Single Precision



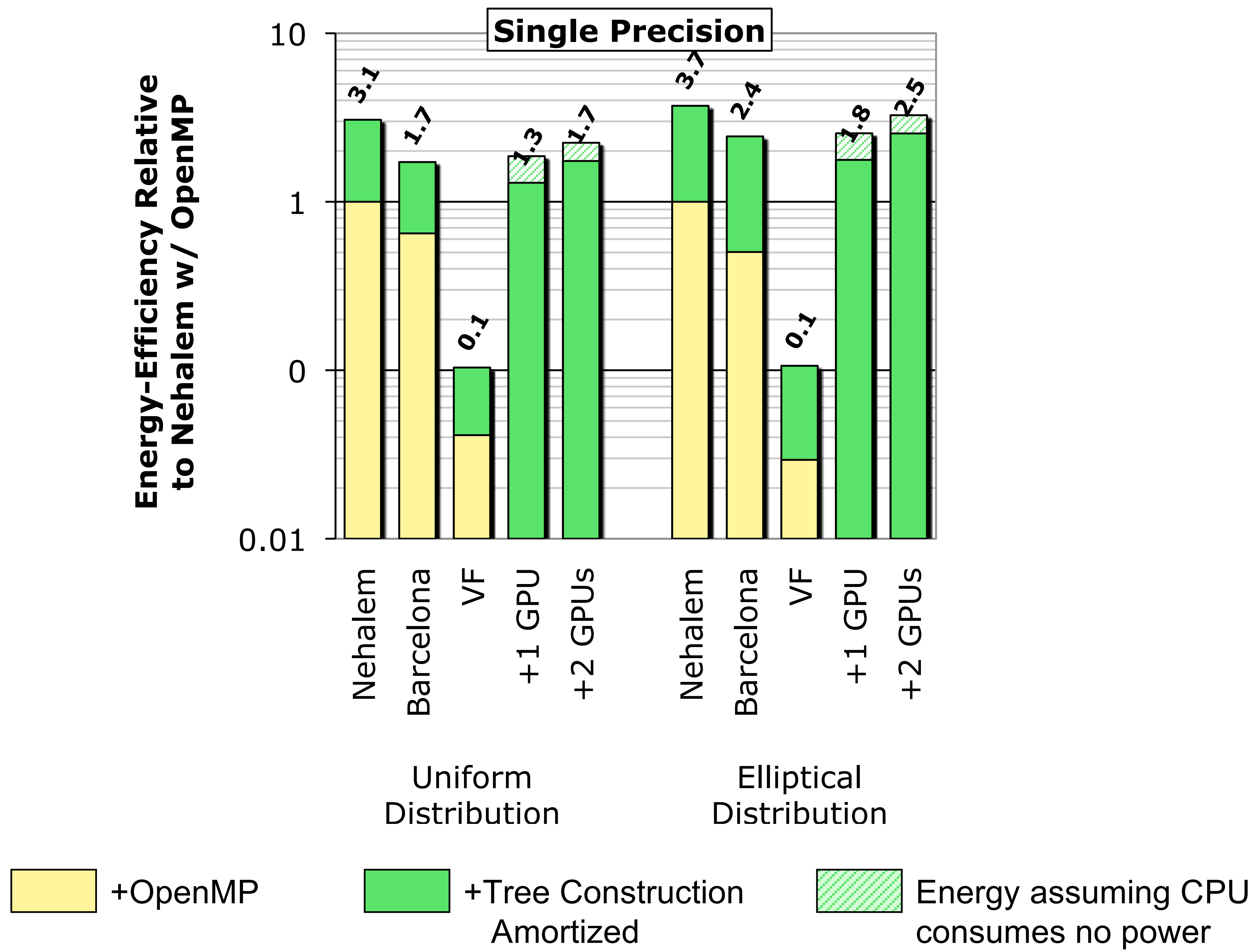
Setup time = time for transforming data to a GPU friendly form.

Transfer time = CPU to GPU transfer time.

Could reduce setup time. But can computation be optimized further?

Cross-Platform Energy-Efficiency Comparison

(Watt-Hours) / (Nehalem+OpenMP Watt-Hours)



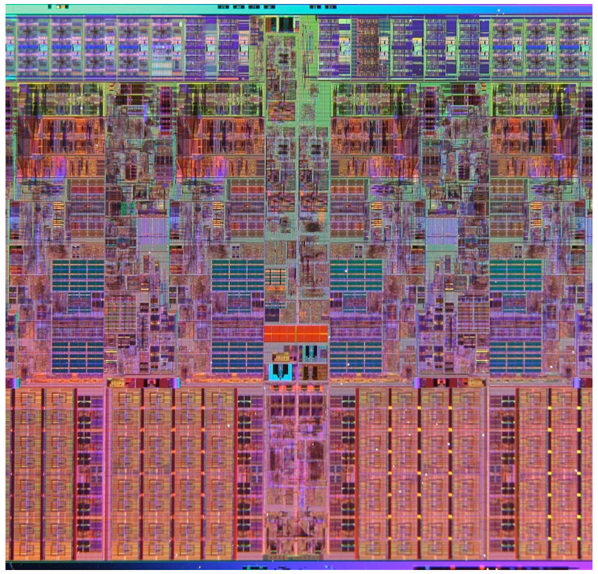
Nehalem has same or better power efficiency than either GPU setup.

Summary and Status

- First extensive multicore platform study for FMM
  - Show 25x Nehalem, 9.4x Barcelona, 37.6x VF from algorithmic, data, and numerical tuning
  - Multicore CPU ~= GPU in power-performance
- Short-term:
  - Perform more detailed modeling → autotuning
  - Build integrated MPI+CPU+GPU implementation
  - Parallel tree construction
- Long-term: Generalize infrastructure and merge with on-going THOR effort for data analysis

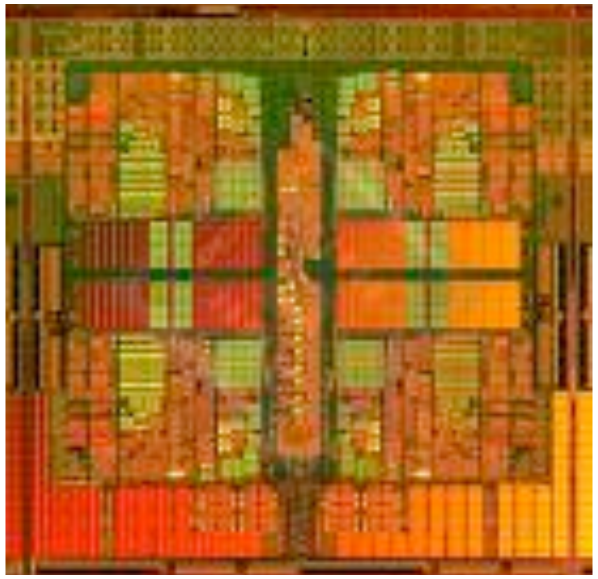
Backup

Memory systems



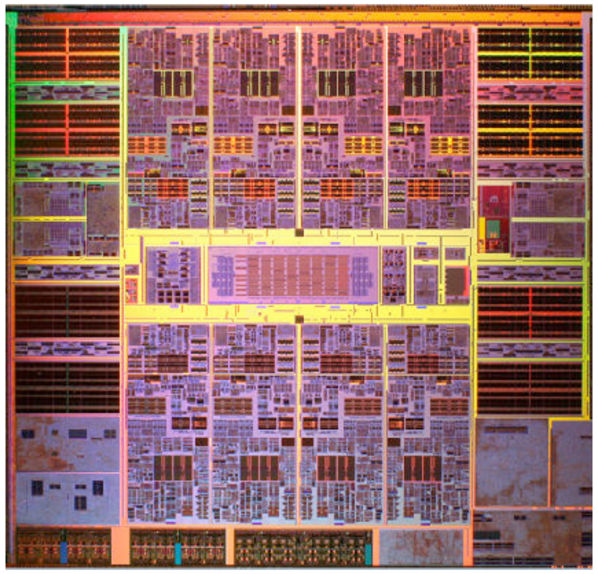
Intel X5550 “Nehalem”

Large (**8 MB**) **L3** cache
High (**51.2 GB/s**) bandwidth



AMD Opteron 2356 “Barcelona”

Smaller (**2 MB**) **L3** cache
Lower (**21.33 GB/s**) bandwidth

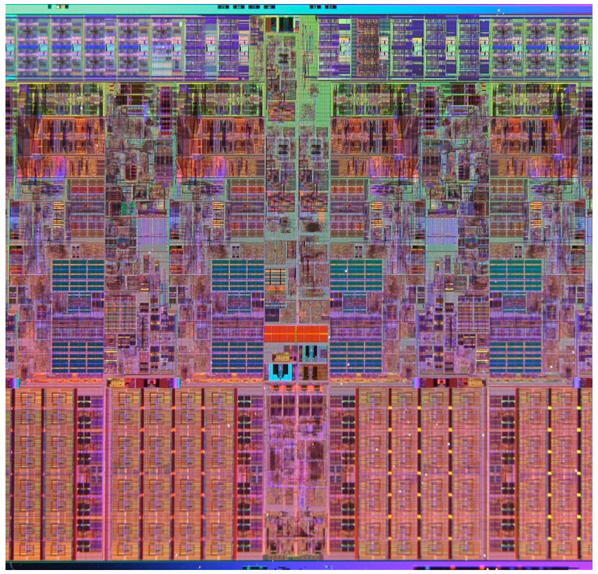


Sun T5140 “Victoria Falls”

4 MB L2
64.0 GB/s bandwidth

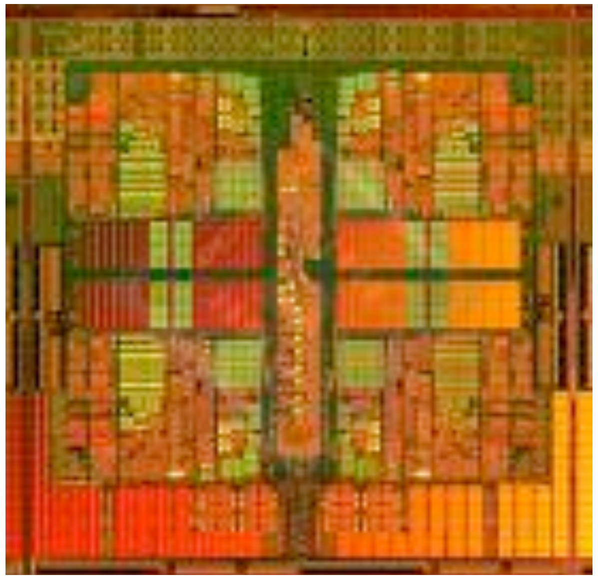
FMM has a mix of memory behaviors, so memory system impact will vary.

SIMD



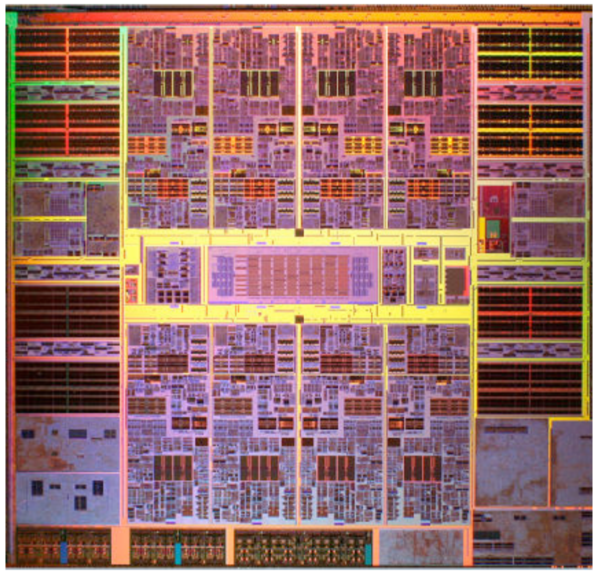
Intel X5550 “Nehalem”

SIMD → 85.5 (double), **170.6** (single) **Gflop/s**



AMD Opteron 2356 “Barcelona”

SIMD → 73.6 (double), **146.2** (single) **Gflop/s**

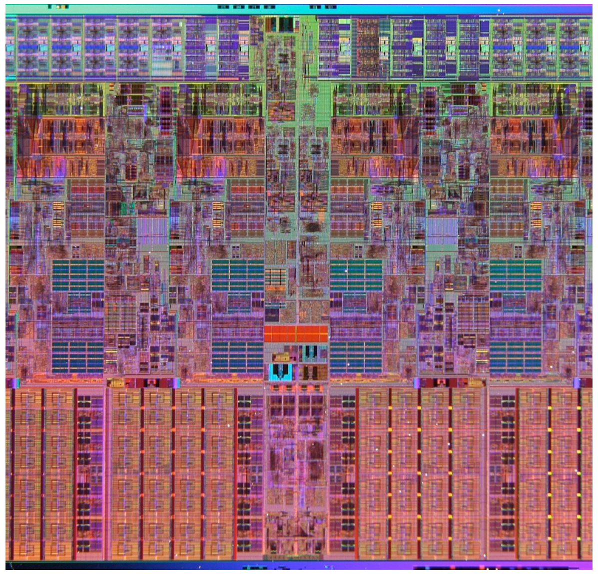


Sun T5140 “Victoria Falls”

No SIMD → 18.66 Gflop/s in single & double

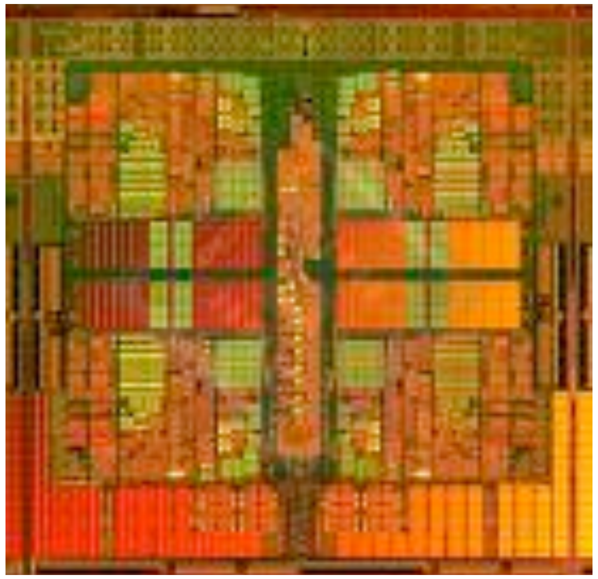
FMM can use SIMD well, so expect good performance on x86.

Floating-point limitations



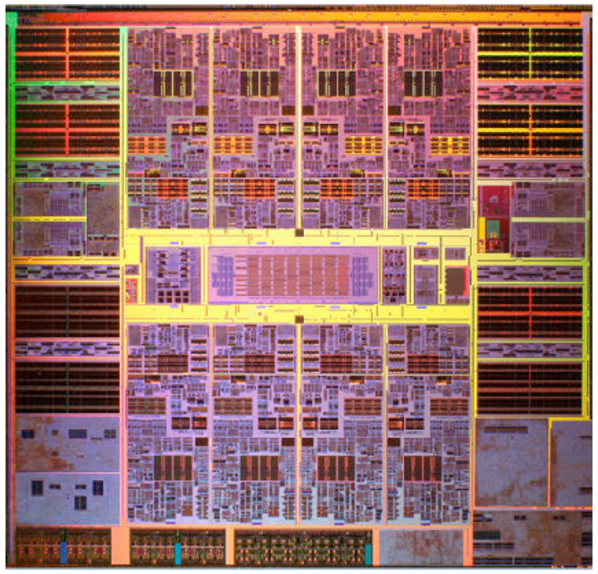
Intel X5550 “Nehalem”

Reciprocal square-root:
0.853 (double), **42.66** (single) **Gflop/s**



AMD Opteron 2356 “Barcelona”

0.897 (double), **73.6** (single) **Gflop/s**

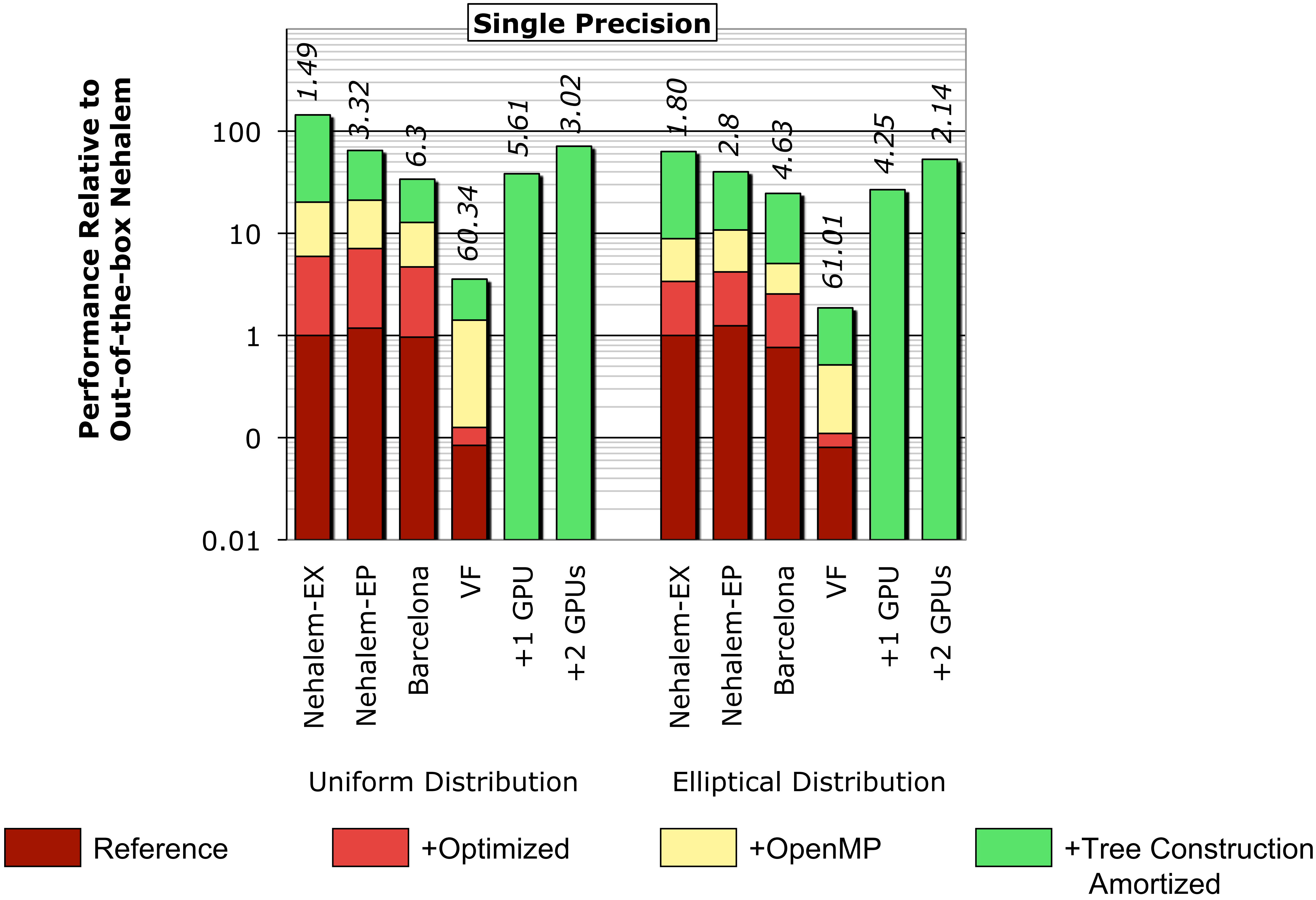


Sun T5140 “Victoria Falls”

2.26 Gflop/s

However, x86 has fast approximate *single-precision* rsqrt*, exploitable in double.*

Cross-Platform Performance Comparison (Summary)



Nehalem-EX outperforms both 1-GPU and 2-GPU case.