Energy Characterization and Optimization of Parallel Prefix-Sums Kernels

Angelos Papatriantafyllou

– Research Group Parallel Computing –

25th August, 2015

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Problem statement

Issues

- Publically available parallel prefix-sums implementations are not energy efficient (SWARM[1], TBB[2], MCSTL[3], Nan_Zhang[4])
- Lack of knowledge about energy-performance trade-off with DVFS and different building-block values

Approach

- Building block optimizations
- Energy characterization in building block level
- Energy characterization of parallel prefix-sums kernels
  - ⋆ building block parameterization and
  - ⋆ DVFS (Dynamic Voltage and Frequency Scaling)

[1]D. A. Bader, V. Kanade, and K. Madduri. "SWARM: A Parallel Programming Framework for Multicore Processors". In: *Proceedings of the 21th International Parallel and Distributed Processing Symposium (IPDPS)*. 2007, pp. 1–8

[2]J. Reinders, ed. *Intel Threading Building Blocks: Oufitting C++ for Multi-core Processor Parallelism*. O'Reilly Media, 2007

[3]J. Singler, P. Sanders, and F. Putze. "MCSTL: The Multi-Core Standard Template Library". In: *Proceedings of the 13th International Euro-Par Conference on Parallel Processing*. 2007, pp. 682–694

[4]Nan Zhang. "A Novel Parallel Scan for Multicore Processors and Its Application in Sparse Matrix-Vector Multiplication". In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 23.3 (2012), pp. 397–404

Contributions

Implementations:

- CPPS (Cache-Aware Parallel Prefix-Sums)
- Optimized sequential prefix-sums kernels (OSPKs)

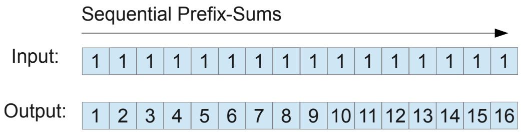
CPPS thorough energy-oriented analysis:

- various different sequential prefix-sums kernels (SPKs)
- performance improvements: 24%-54% (different OSPKs and number of threads)
- energy savings: 24%-55% (lower CPU frequencies)
- different thread placements (TPPs) over NUMA architectures

Prefix-sums problem

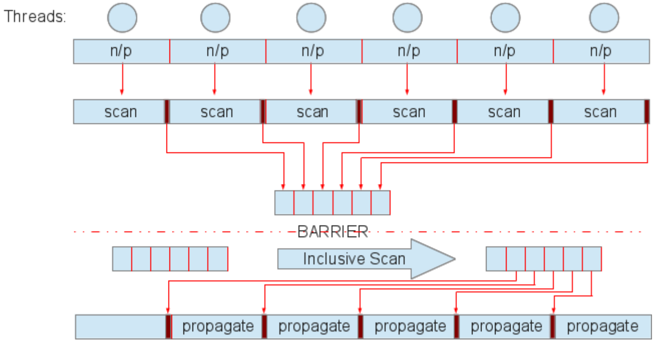
A memory bound kernel (Prefix-sums)

Definition: Given an input array x of n elements, the problem is to compute the *(inclusive) prefix-sums* $\oplus_{j=0}^{i} x[j]$, for all indices i, $0 \le i < n$, for a given, associative operation $\oplus$. The prefix-sums will be computed *in-place* with ith prefix-sum stored in $x[i]$



Applicable to: Load balancing, book keeping, solving recurrences, compacting arrays, and many more

Prefix-sum: Parallel approach



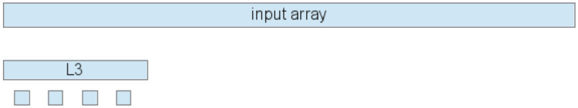
Current state-of-the-art parallel approaches (MCSTL, TBB, Nan Zhang, SWARM, also many more on GPUS)

Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3

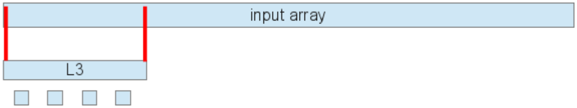
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



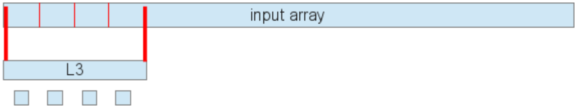
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



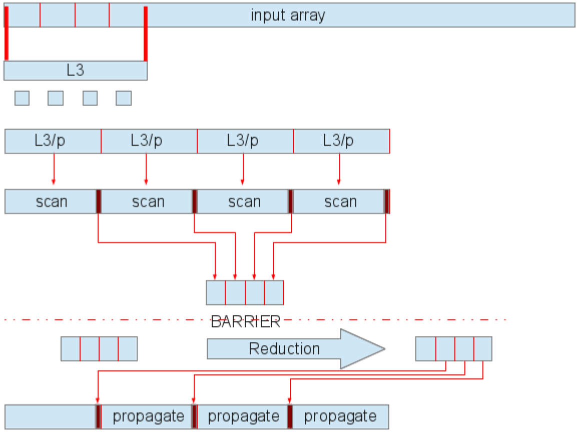
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



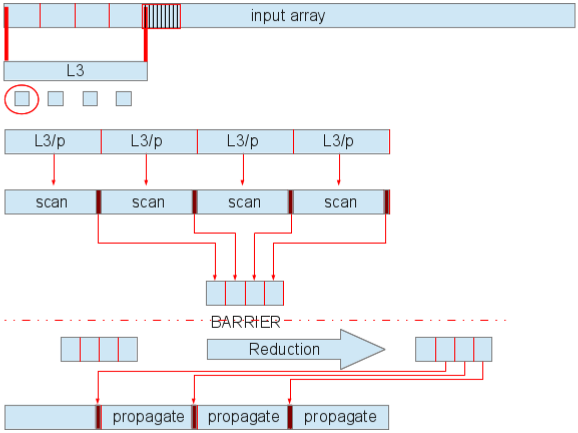
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



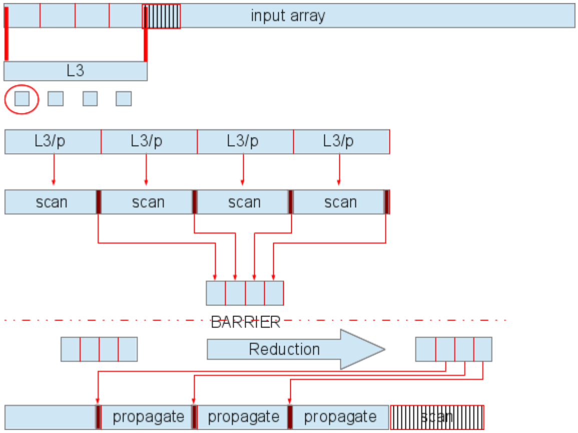
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



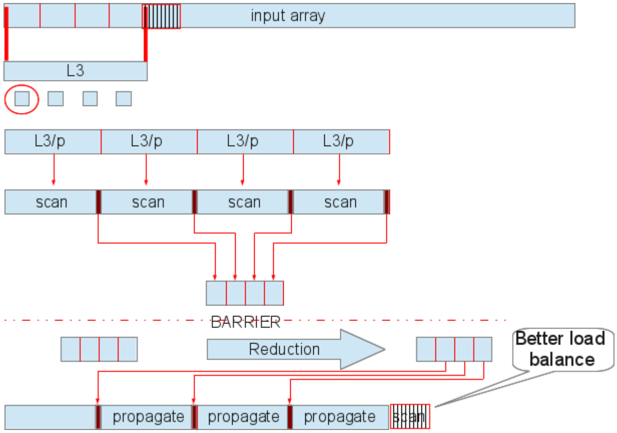
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



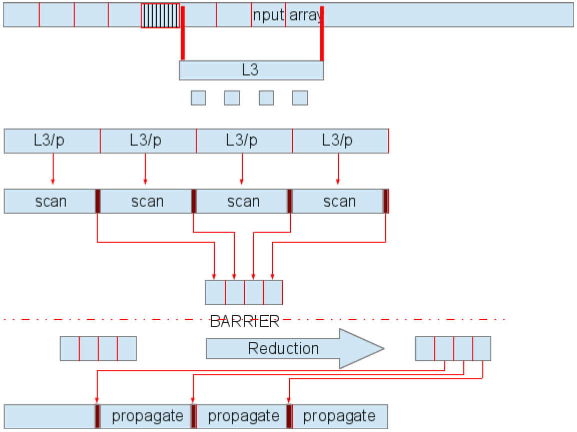
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



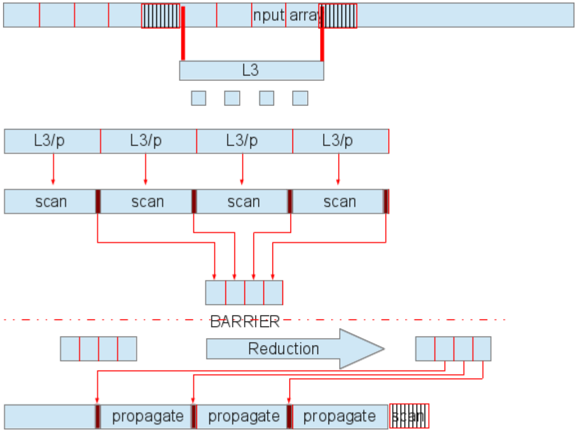
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



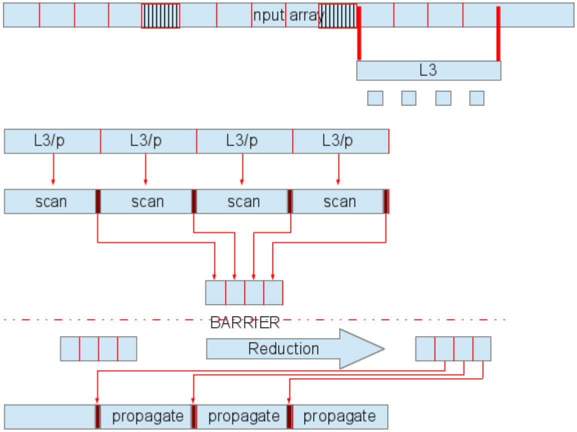
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



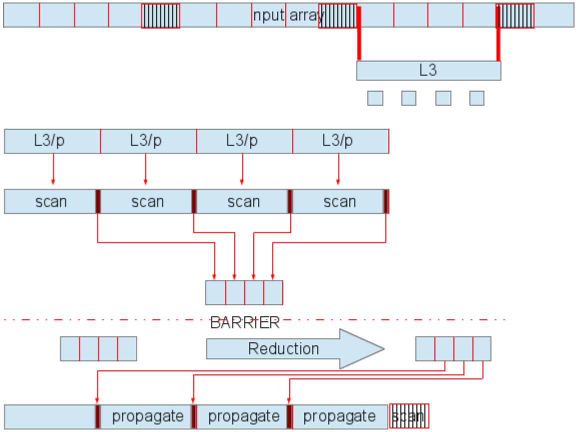
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



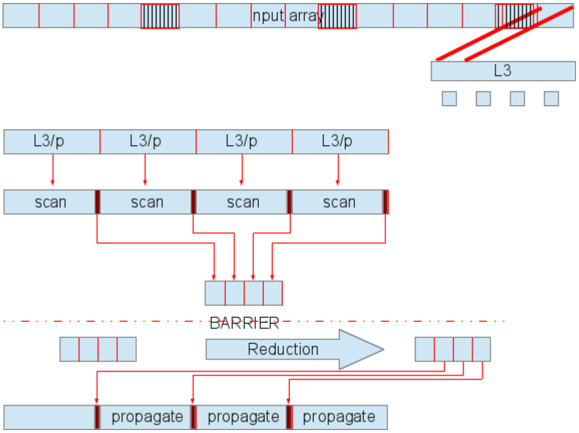
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



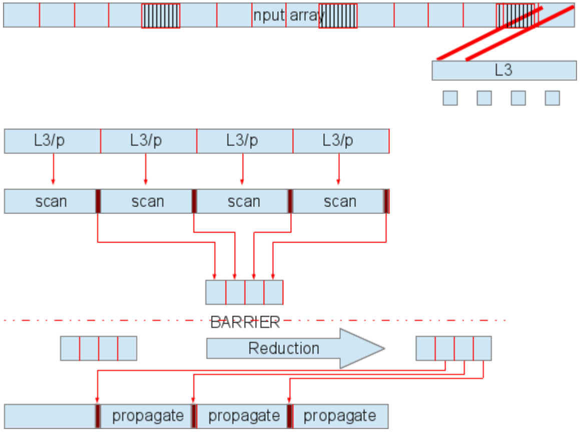
Our approach: CPPS

- Cache aware and load balanced
- Example: 4 threads (sharing L3 cache) perform the prefix-sum in an array with size 4×L3



Our approach: CPPS

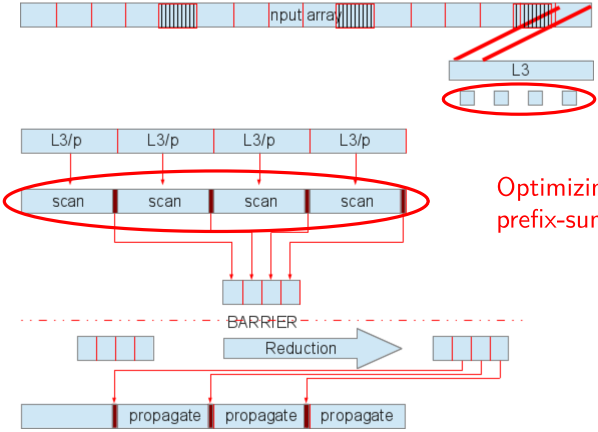
Can we optimize it more?



Our approach: CPPS

Can we optimize it more?

Yes



Place threads across NUMA nodes

Optimizing the sequential prefix-sums kernel

Trivial sequential prefix-sums kernel

Trivial_seq ($O(n)$):

- used by most parallel implementations
- exception: Intel's SHOC benchmark suite[5]

```
trivial_seq(x[], n) {
  for(i=1; i<n; i++) {
    x[i] = x[i] OP x[i-1];
  }
}
```

Drawbacks:

- only data dependent operations
- significant amount of processor stalls
- not the most energy/performance efficient prefix-sums kernel
- provide poor performance and increase energy consumption of CPPS

[5] R. Rahman. *The Scalable Heterogeneous Computing Benchmark Suite SHOC for Intel® Xeon Phi™*. https://software.intel.com/en-us/blogs/2013/03/20/the-scalable-heterogeneous-computing-benchmark-suite-shoc-for-intelr-xeon-phitm. 2013.

Optimized sequential prefix sums kernels

Three algorithms:

- seq1 exploits vectorization instructions ($O(2n)$)
- seq2 works for non-vectorized datatypes ($O(2n)$)
- seq3 better loop unrolling ($O(n)$)

Optimized sequential prefix sums kernels

Three algorithms:

- seq1 exploits vectorization instructions ($O(2n)$)
- seq2 works for non-vectorized datatypes ($O(2n)$)
- seq3 better loop unrolling ($O(n)$)

Seq1 and seq2:

- are based on[6]for vector processors
- break data dependencies (better pipelining)
- perform independent prefix-sums operations
- chunk input array for cache reusability improvements
- parameter A: input array chunk size
- parameter V: number of independent prefix-sums operations

Optimized sequential prefix sums kernels

Three algorithms:

- seq1 exploits vectorization instructions ($O(2n)$)
- seq2 works for non-vectorized datatypes ($O(2n)$)
- seq3 better loop unrolling ($O(n)$)

Seq1 and seq2:

- are based on[7]for vector processors
- break data dependencies (better pipelining)
- perform independent prefix-sums operations
- chunk input array for cache reusability improvements
- parameter A: input array chunk size
- parameter V: number of independent prefix-sums operations

Seq3:

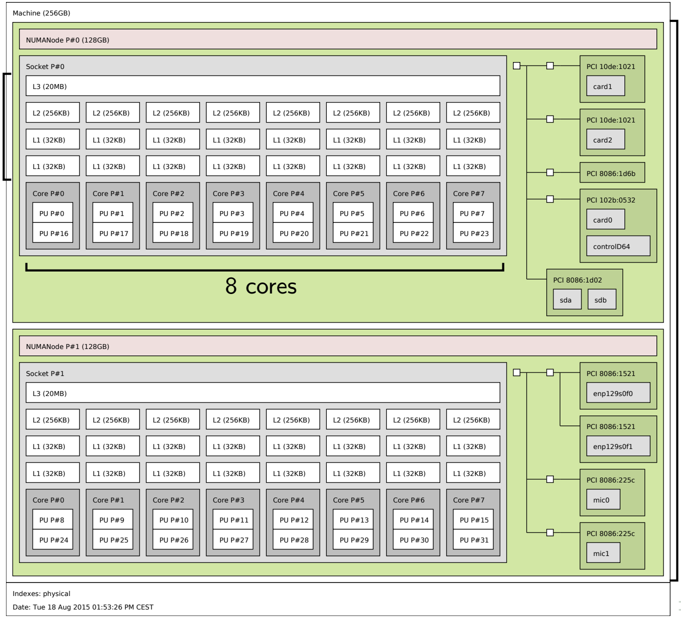
- derives from trivial_seq
- two level nested loop
- parameter k: depth of inner loop
- parameter V: iteration space

Open questions

- Is CPPS a high performance parallel prefix-sums kernel?
- Can an OSPK improve CPPS performance?
- How much OSPKs improve CPPS performance and energy?
- What is CPPS performance-energy trade-off for NUMA-aware thread placement?
- What is CPPS performance/energy behavior in different frequency levels?

Experimental setup (Machine)

3 cache levels



8 cores

2 NUMA nodes
16 cores

Experimental setup - Energy reports

Energy reports through Intel's RAPL (Run Average Power Limit):

- estimates power and energy consumption (accurately[8])
- control through special hardware registers (MSRs)
- contains 4 distinct domains:
  - ⋆ PKG: whole CPU package (cores+uncore) (Desktop/Server version)
  - ⋆ PP0: Processor cores only (Desktop/Server version)
  - ⋆ PP1: A specific device in the uncore (Desktop version)
  - ⋆ DRAM: Memory controller (Server version)

[8] M. Hähnel et al. "Measuring Energy Consumption for Short Code Paths Using RAPL". In: *SIGMETRICS Performance Evaluation Review* 40.3 (2012), pp. 13–17.

Experimental setup - Energy reports

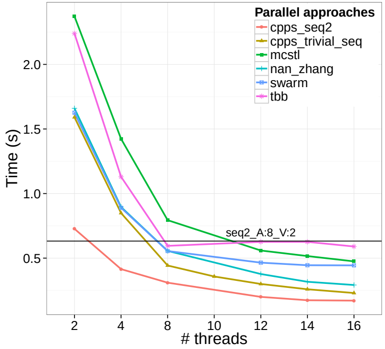
RAPL domain usage:

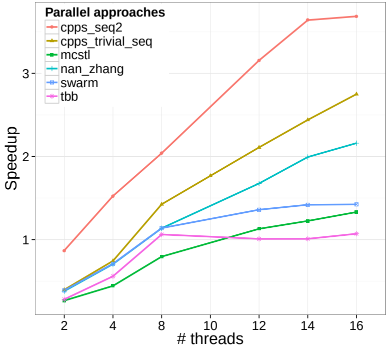
- PKG when running at maximum frequency
- PKG avoided when DVFS is used
- the uncore consumes constant power according to[9]
- PP0 when monitoring DVFS effect
- DRAM was not used: cannot be affected by DVFS

[9] B. Subramaniam and W. Feng. “Towards Energy-Proportional Computing for Enterprise Class Server Workloads”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*. 2013, pp. 15–26.

CPPS Vs (MCSTL, TBB, ...)

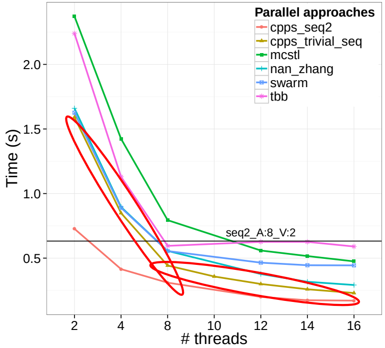
Problem size: 10^9 32-bit integers, TPP: Compact, Frequency: 2.6 GHz (max)

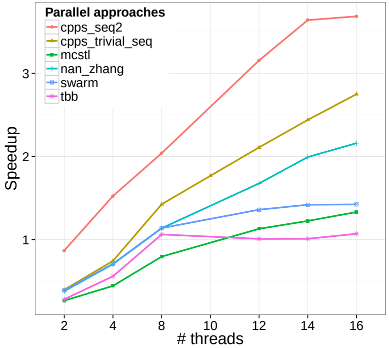




CPPS Vs (MCSTL, TBB, ...)

Problem size: 10^9 32-bit integers, TPP: Compact, Frequency: 2.6 GHz (max)

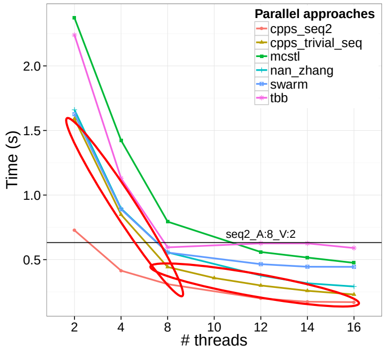


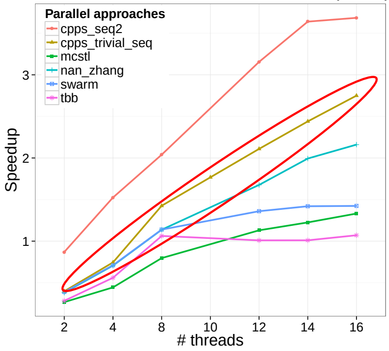


- CPPS outperforms all tested approaches (better cache reusability and load balancing)

CPPS Vs (MCSTL, TBB, ...)

Problem size: 10^9 32-bit integers, TPP: Compact, Frequency: 2.6 GHz (max)

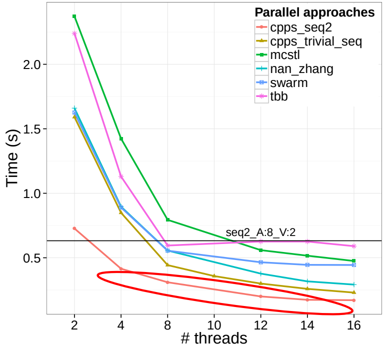


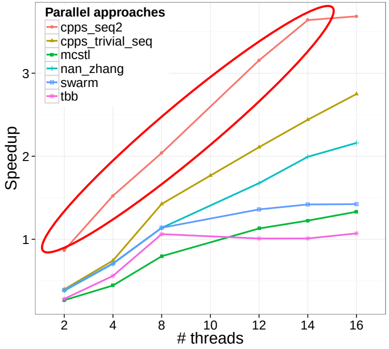


- CPPS outperforms all tested approaches (better cache reusability and load balancing)
- CPPS exhibits better scalability

CPPS Vs (MCSTL, TBB, ...)

Problem size: 10^9 32-bit integers, TPP: Compact, Frequency: 2.6 GHz (max)





- CPPS outperforms all tested approaches (better cache reusability and load balancing)
- CPPS exhibits better scalability
- CPPS performs and scales even better with optimized kernel
- All other approaches use trivial_seq

Thread placement policies

CPPS scalability issues:

- CPPS performance influenced by memory bandwidth
- CPPS needs memory bandwidth
- more memory bandwidth comes with more NUMA nodes
- by avoiding sockets memory bandwidth saturation
- possible solution: NUMA-aware thread placement

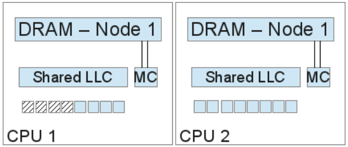
Thread placement policies

CPPS scalability issues:

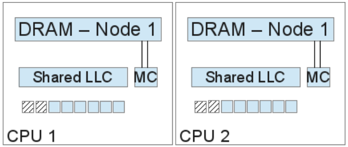
- CPPS poor scalability mostly is based on memory bandwidth saturation
- CPPS needs memory bandwidth
- more memory bandwidth comes with more NUMA nodes and
- by avoiding memory bandwidth saturation
- possible solution: NUMA-aware thread placement

Two thread placement policies (TPPs):

- Compact



- Scatter



Performance-energy trade-off on different TPPs

- We explore CPPS performance-energy trade-off on different thread placements (TPPs) and
- with different optimized sequential kernels (OSPKs)
- Compact or Scatter?
- Can Scatter be favored by an OSPK?
- We know by activating a second socket results in double increase in energy
- We do not know the performance improvements

Performance-energy trade-off on different TPPs

- CPPS has been configured with all combinations resulted by two sets:
  - ⋆ sequential kernel: {`seq2_A:128-V:16`, `trivial_seq`}
  - ⋆ thread placements: {`Compact`, `Scatter`}
- performance and energy comparison between these CPPS configurations
- energy estimations based on RAPL's PKG

Problem size: 10^9 32-bit integers, Frequency: 2.6 GHz (max)

Sequential kernel	# threads	Gain (%) Scatter to Compact	
		Energy	Time
seq2_A:128_V:16	2	46.86	-0.08
trivial_seq	2	47.50	0.39
seq2_A:128_V:16	4	45.56	-0.36
trivial_seq	4	45.75	0.15
seq2_A:128_V:16	8	42.21	-3.80
trivial_seq	8	43.62	-0.59
seq2_A:128_V:16	12	-0.85	-0.48
trivial_seq	12	1.81	0.12
seq2_A:128_V:16	16	0	0
trivial_seq	16	0	0

Performance-energy trade-off on different TPPs

- CPPS has been configured with all combinations resulted by two sets:
  - ⋆ sequential kernel: {`seq2_A:128-V:16`, `trivial_seq`}
  - ⋆ thread placements: {`Compact`, `Scatter`}
- performance and energy comparison between these CPPS configurations
- energy estimations based on RAPL's PKG

Problem size: 10^9 32-bit integers, Frequency: 2.6 GHz (max)

Sequential kernel	# threads	Gain (%) Scatter to Compact	
		Energy	Time
seq2_A:128_V:16	2	46.86	-0.08
trivial_seq	2	47.50	0.39
seq2_A:128_V:16	4	45.56	-0.36
trivial_seq	4	45.75	0.15
seq2_A:128_V:16	8	42.21	-3.80
trivial_seq	8	43.62	-0.59
seq2_A:128_V:16	12	-0.85	-0.48
trivial_seq	12	1.81	0.12
seq2_A:128_V:16	16	0	0
trivial_seq	16	0	0

Negligible performance improvement

Performance-energy trade-off on different TPPs

- CPPS has been configured with all combinations resulted by two sets:
  - ⋆ sequential kernel: {`seq2_A:128-V:16`, `trivial_seq`}
  - ⋆ thread placements: {`Compact`, `Scatter`}
- performance and energy comparison between these CPPS configurations
- energy estimations based on RAPL's PKG

Problem size: 10^9 32-bit integers, Frequency: 2.6 GHz (max)

Sequential kernel	# threads	Gain (%) Scatter to Compact	
		Energy	Time
seq2_A:128_V:16	2	46.86	-0.08
trivial_seq	2	47.50	0.39
seq2_A:128_V:16	4	45.56	-0.36
trivial_seq	4	45.75	0.15
seq2_A:128_V:16	8	42.21	-3.80
trivial_seq	8	43.62	-0.59
seq2_A:128_V:16	12	-0.85	-0.48
trivial_seq	12	1.81	0.12
seq2_A:128_V:16	16	0	0
trivial_seq	16	0	0

Negligible performance improvement

Double energy loss

Performance-energy trade-off on different TPPs

- CPPS has been configured with all combinations resulted by two sets:
  - ⋆ sequential kernel: {`seq2_A:128-V:16`, `trivial_seq`}
  - ⋆ thread placements: {`Compact`, `Scatter`}
- performance and energy comparison between these CPPS configurations
- energy estimations based on RAPL's PKG

Problem size: 10^9 32-bit integers, Frequency: 2.6 GHz (max)

Sequential kernel	# threads	Gain (%) Scatter to Compact	
		Energy	Time
seq2_A:128_V:16	2	46.86	-0.08
trivial_seq	2	47.50	0.39
seq2_A:128_V:16	4	45.56	-0.36
trivial_seq	4	45.75	0.15
seq2_A:128_V:16	8	42.21	-3.80
trivial_seq	8	43.62	-0.59
seq2_A:128_V:16	12	-0.85	-0.48
trivial_seq	12	1.81	0.12
seq2_A:128_V:16	16	0	0
trivial_seq	16	0	0

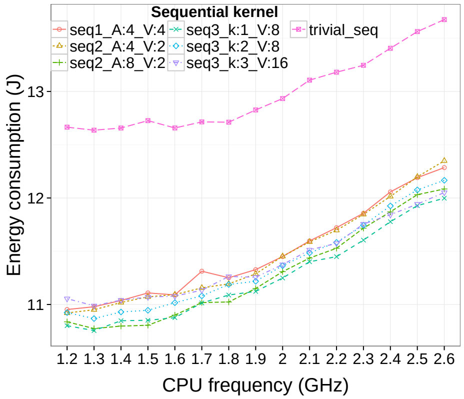
OSPK and SCATTER does not favor CPPS in performance

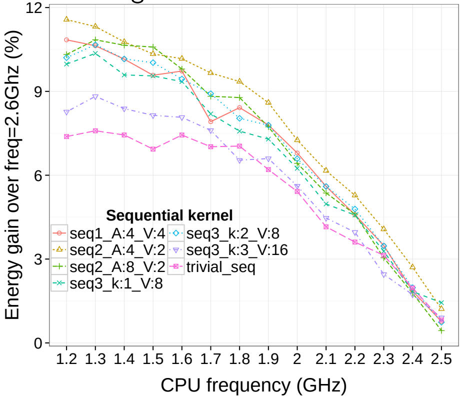
Negligible performance improvement

Double energy loss

Optimized vs non-optimized sequential prefix-sums kernels

Problem size: 10^9 32-bit integers

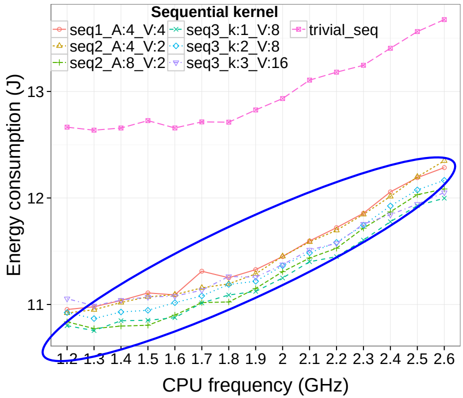


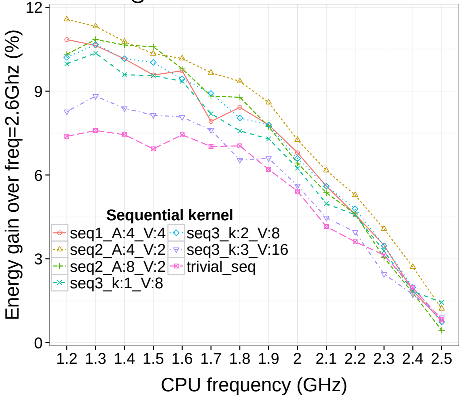


- energy consumption results for 15 different CPU frequencies
- 6 optimized vs one non-optimized sequential prefix-sums
- energy estimations based on RAPL's PP0

Optimized vs non-optimized sequential prefix-sums kernels

Problem size: 10^9 32-bit integers

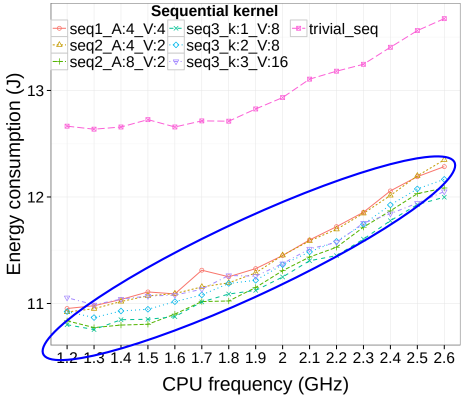


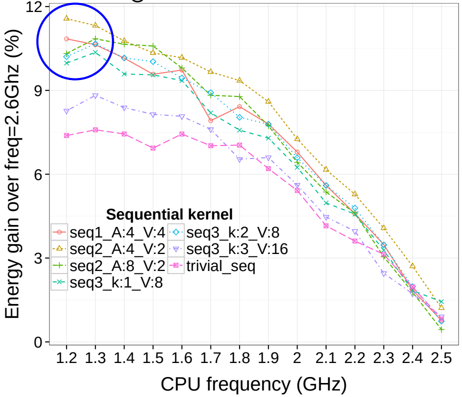


- energy consumption results for 15 different CPU frequencies
- 6 optimized vs one non-optimized sequential prefix-sums
- energy estimations based on RAPL's PP0
- all optimized kernels reduce energy consumption (left figure)

Optimized vs non-optimized sequential prefix-sums kernels

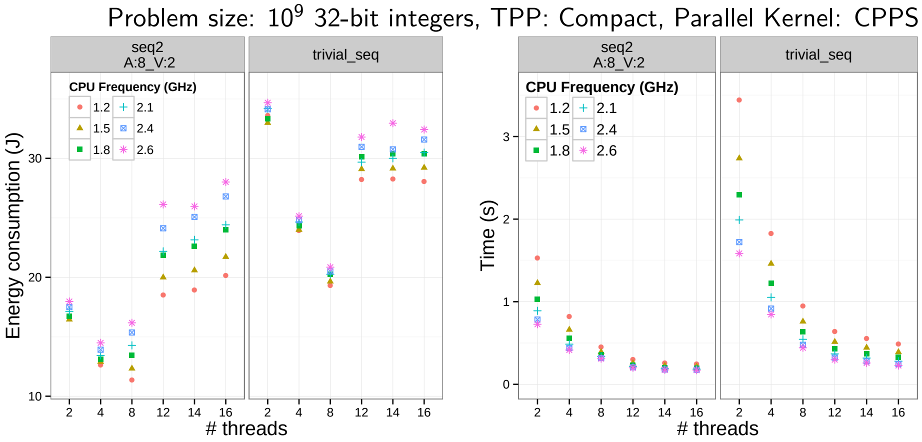
Problem size: 10^9 32-bit integers





- energy consumption results for 15 different CPU frequencies
- 6 optimized vs one non-optimized sequential prefix-sums
- energy estimations based on RAPL's PP0
- all optimized kernels reduce energy consumption (left figure)
- some kernels have better energy scalability while descending the frequency levels (right figure)

CPPS+DVFS+Sequential_prefix_sums

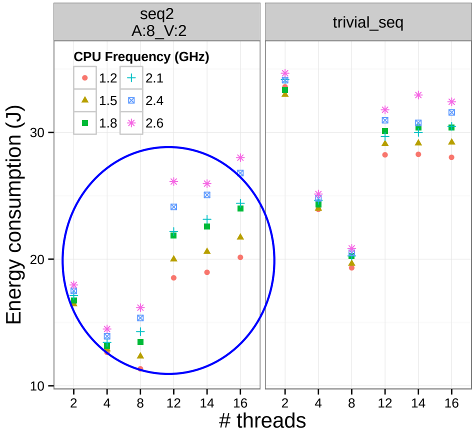


CPPS performance and energy results when configured with:

- 6 different number of threads (2,4,8,12,14,16 threads)
- 4 optimized and 1 non-optimized sequential prefix-sums kernels
- 6 different CPU frequencies (1.2, 1.5, 1.8, 2.1, 2.4, 2.6 GHz)
- energy estimations based on RAPL's PP0

CPPS+DVFS+Sequential_prefix_sums

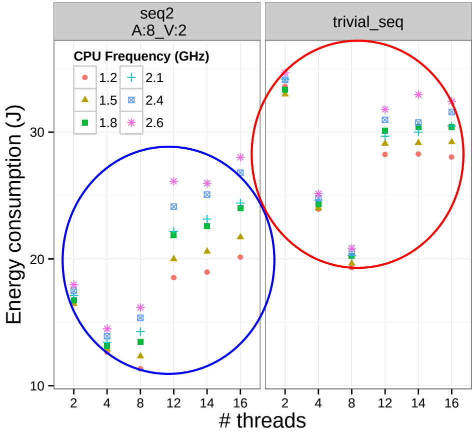
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- CPPS reduces energy consumption independent of number of threads and frequency when configured with optimized sequential prefix-sums kernels
- energy savings 24%-55%

CPPS+DVFS+Sequential_prefix_sums

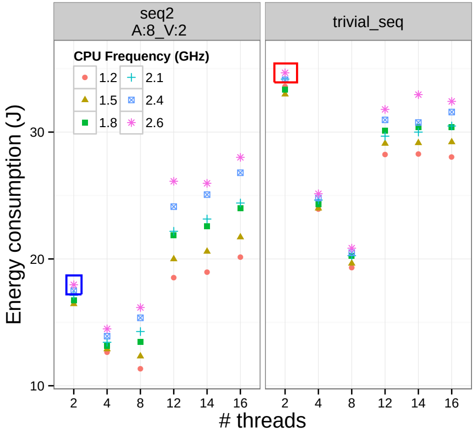
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- CPPS reduces energy consumption independent of number of threads and frequency when configured with optimized sequential prefix-sums kernels
- energy savings 24%-55%

CPPS+DVFS+Sequential_prefix_sums

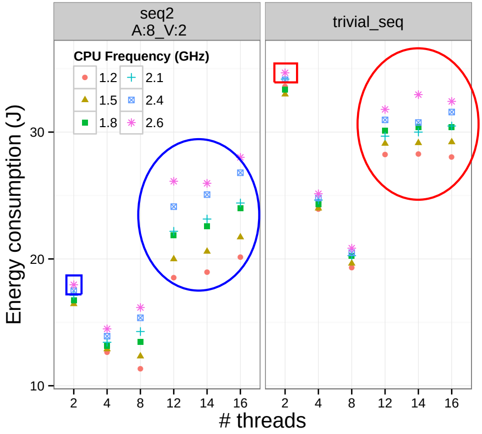
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- CPPS reduces energy consumption independent of number of threads and frequency when configured with optimized sequential prefix-sums kernels
- energy savings 24%-55%
- threads: 2, SPK: seq2_A:8_V:2, frequency 2.4 GHz $\rightarrow$ 35% less energy

CPPS+DVFS+Sequential_prefix_sums

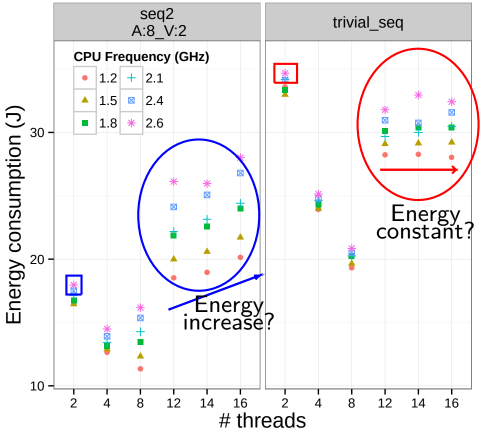
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- CPPS reduces energy consumption independent of number of threads and frequency when configured with optimized sequential prefix-sums kernels
- energy savings 24%-55%
- threads: 2, SPK: seq2_A:8_V:2, frequency 2.4 GHz $\rightarrow$ 35% less energy

CPPS+DVFS+Sequential_prefix_sums

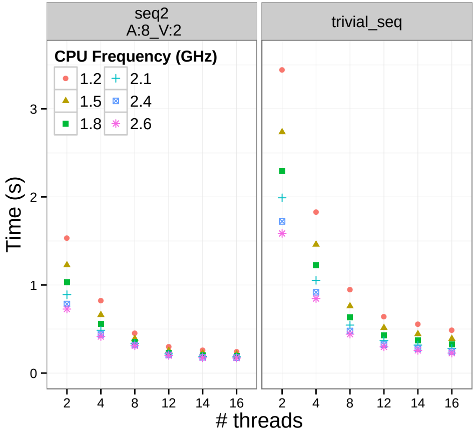
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- CPPS reduces energy consumption independent of number of threads and frequency when configured with optimized sequential prefix-sums kernels
- energy savings 24%-55%
- threads: 2, SPK: seq2_A:8_V:2, frequency 2.4 GHz → 35% less energy

CPPS+DVFS+Sequential_prefix_sums

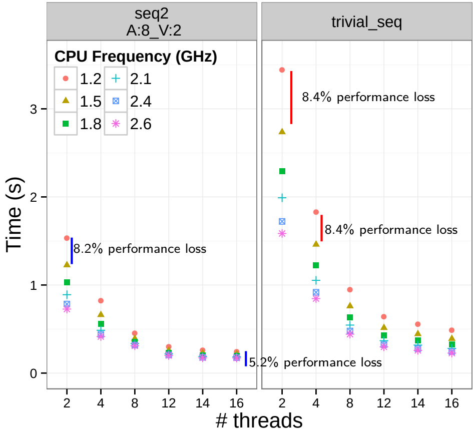
Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- lower the frequency → higher the performance loss

CPPS+DVFS+Sequential_prefix_sums

Problem size: 10^9 32-bit integers, TPP: Compact, Parallel Kernel: CPPS



- lower the frequency → higher the performance loss
- CPPS performance penalization rate increases with optimized kernels (3.7%-8.2%)
- but reduces with more threads (i.e., 16 threads: 1.3%-5.2%)
- instead with trivial_seq increases 4%-8.4% for every number of thread

Conclusion

We have seen:

- energy efficient optimized sequential prefix-sums kernels (OSPK)
- sequential kernel is a dominant building block
- CPPS significant energy savings with OSPKs
- less CPPS performance loss
- no impact of TPPs to OSPKs behavior
- TPP under consideration
- CPPS energy-performance optimization: a multivariable optimization problem

Future work:

- investigate CPPS performance/energy behavior on larger shared-memory systems and
- on Intel's XEON Phi coprocessor
- plug CPPS as a building block into other algorithms (summed-area table and stream compaction)