INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

Processor-Aware Cache-Oblivious (PACO) Algorithms

Yuan Tang[1], Weiguo Gao[2]

[1]School of Computer Science and School of Software,

[2]School of Mathematical Sciences and School of Data Science

Fudan University

Motivations:

- Frigo et al. proposed an ideal cache model and a recursive technique to design sequential cache-efficient algorithm on a hierarchical architecture of caches in a cache-oblivious fashion
- Two Open Problems:
  - ***Portability***: How to extend the technique to an ***arbitrary*** architecture?
  - ***Scalability***: How to run an algorithm ***exactly*** and ***efficiently*** on an ***arbitrary*** number of processors, e.g. Strassen's algorithm
    - Attaining computation lower bound exactly
    - Attaining communication lower bound up to a constant factor

Two Classic Ways of Extension:

- Processor-Oblivious (PO):
  - Only specifies data dependency, and leave an efficient runtime parallelization to a runtime scheduler or folding mechanism (Network-Oblivious)
  - Inputs: Sequential Cache Complexity, Critical-Path Length
  - Machinery: A Runtime Scheduler (shared-memory) or Folding Mechanism (distributed-memory)
  - Output: Parallel Complexity Bounds
  - Main Benefits:
    - Easy-of-programming, more adaptive to non-dedicated settings, scalable to an arbitrary number of processors (within a certain range)
  - Main Concerns:
    - Usually more communication overheads than a PA counterpart
    - May require to choose a proper base-case size
    - !! Fundamentally due to the runtime scheduler has no knowledge of the algo

Two Classic Ways of Extension:

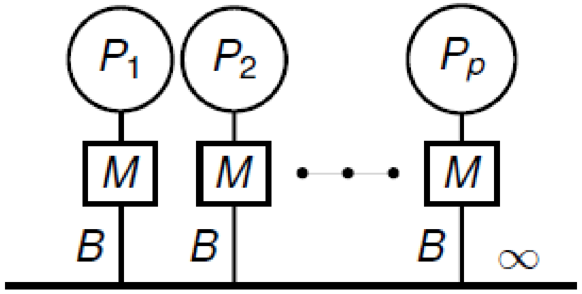
- Processor-Aware (PA):
  - Calculate (work and time) complexity bounds along a critical path
  - Main Benefits:
    - Better performance in both theory and practice
  - Main Concerns:
    - A 2D, 2.5D, or 3D MM algorithm may require p to be factorizable into two or three roughly equal numbers
    - The CAPS Strassen's algorithm requires that p is an exact power of 7
    - Lipshitz et al. later improved p to be a multiple of 7 with no large prime factors, i.e. $p = m \cdot 7^x$, where $1 \leq m < 7$ and $1 \leq x$ are integer numbers, by a hybrid of Strassen and classic $O(n^3)$ MM algorithms

Contributions:

- A novel Processor-Aware but Cache-Oblivious (PACO) way of partitioning a cache-oblivious algorithm to achieve perfect strong scaling based on a pruned BFS traversal of the algorithm's divide-and-conquer tree
- The applications include, but not limited to, LCS, 1D, GAP, MM, Strassen, TRS, Cholesky, LUPP, QR, and Comparison-Based Sorting
- Works on shared-memory, distributed-memory, and heterogeneous computing systems
- Provides an almost exact solution to the ***Open Problem*** on ``Parallelizing Strassen exactly and efficiently on an arbitrary number of processors''
- Provides a new perspective on the ***Open Problem*** on ``Exten recursive cache-oblivious technique to an arbitrary architectu

Models:

- Computational Model: DAG
  - Vertex: a piece of computation with no parallel construct. Each arithmetic operation is an $O(1)$ operation
  - Edge: data dependency
- Machine Model: The ``ideal distributed-cache model''
  - p dedicated processors with identical performance
  - Two-level memory model
  - Non-interfering private caches: cache misses of each processors can be analyzed independently
    - Valid under the DAG-consistent memory model maintained by the Backer protocol or the HSMS model
  - We do not consider cache-coherence protocol or false sharing. Non
    algorithms have data race.

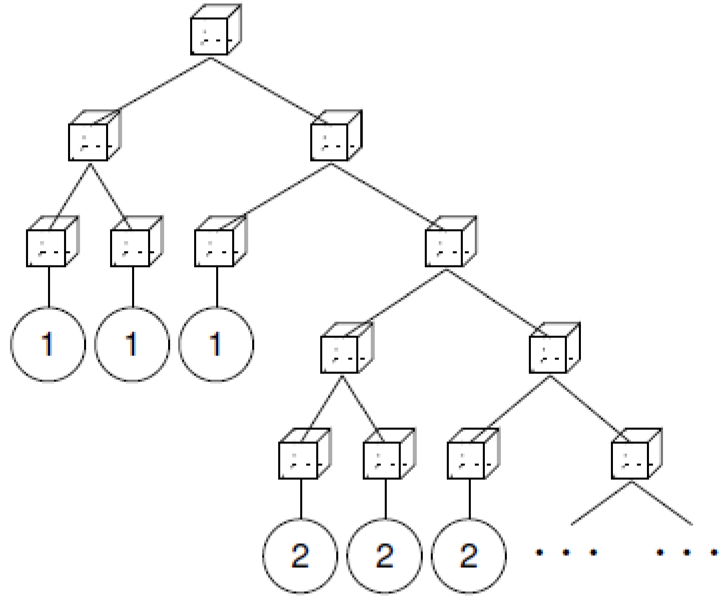


Ideal Distributed-Cache Model (Frigo and Strumpen)

A General PACO Algorithm:

- A sequential cache-oblivious algorithm can be viewed as a DFS traversal of the algorithm's divide-and-conquer tree
- A parallel one is usually some interleaving of DFS and BFS traversal of the same tree
  - DFS: reduce memory footprint
  - BFS: increase parallelism
- A PACO algorithm is a ***pruned BFS*** traversal of the same tree
  - Basic observation: the maximal speedup attainable on a p-processor system is usually p-fold

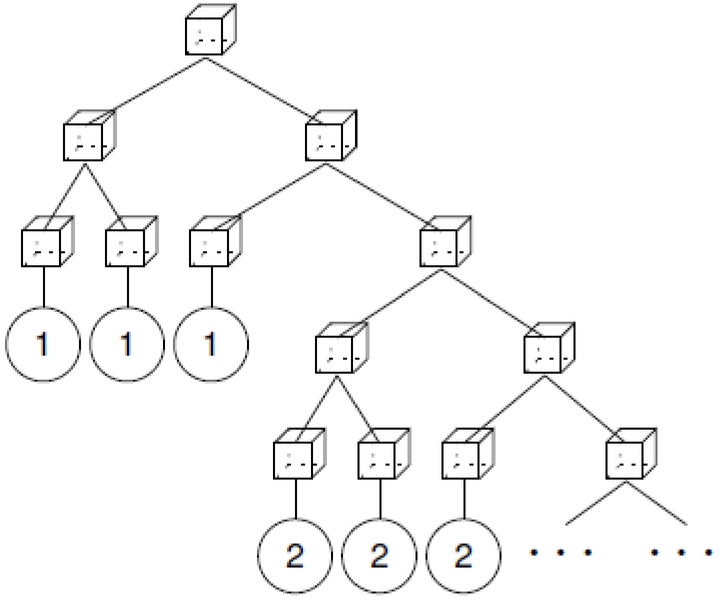
  1. Unfolds the algorithm's c-way divide-and-conquer tree depth-by-depth in a BFS fashion
  2. If some depth has $\geq p$ ``***ready***'' nodes, up to $(c-1) \cdot p$ of them will be pruned, i.e. will be executed sequentially (DFS) on the assigned processor
  3. The rest of nodes will go to more rounds
  4. The procedure repeats until either all pruned, or base cases.



Pruned BFS traversal of a binary tree (p=3)
Labels indicate assigned (pruned) order

A General PACO Algorithm:

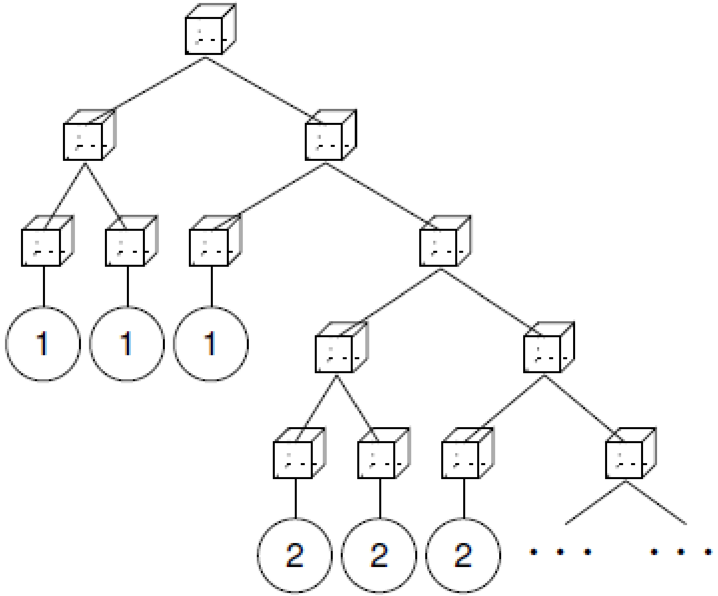
- A PACO algorithm is a pruned BFS traversal of the same tree:
    - ***Invariants***:
        - The node(s) are partitioned evenly and exactly into p sets
        - Each set forms a geometrically decreasing sequence in both ***comp***. and ***comm***.
        - Top-level node(s) dominate
    - Classic PO: divides each and every node to base cases to increase the ``slackness''
        - More slackness means more potential deviations from sequential execution order, thus more comm. And sync. overheads
    - Classic PA: constrained by the structure of algorithm
        - The CAPS Strassen's algorithm: p must be an exact power of 7
        - Later improved to a multiple of 7 with no large prime factors by a hybrid of Strassen and classic MM
        - 2D, 2.5D, 3D MM algorithms require p to be factorizable into 2 or 3 roughly equal numbers



Pruned BFS traversal of a binary tree (p=3)
Labels indicate assigned (pruned) order

Complexity Counting:

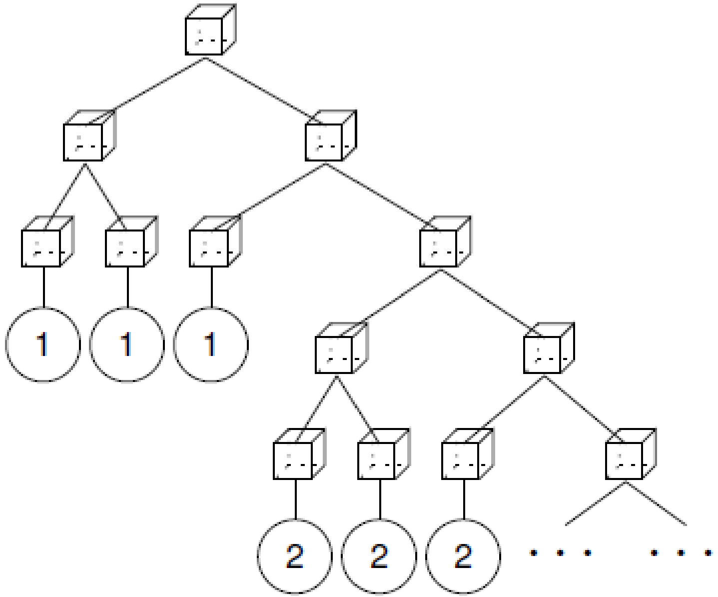
- Non-Interfering Caches: all nodes (tasks) are counted independently
- Perfect Strong Scaling Property (initiated by Ballard et al.):
  - ***Optimal balanced comp***. and ***Optimal balanced comm.***:
    - The ***overall*** T_p^{Σ} and Q_p^{Σ} be ***asymptotically optimal***
    - The quantity along a critical path, i.e. $T_p^{\max} = \left(\frac{1}{p}\right) T_p^{\Sigma}$ and $Q_p^{max} = \left(\frac{1}{p}\right) Q_p^{\Sigma}$
    - The ***difference*** can not be more than an ***asymptotically smaller*** term
    - Be ***valid*** for ***an arbitrary number*** of processors
  - Classic PO: counts only sequential cache and critical-path length, relies on a runtime scheduler (e.g. RWS) or a folding mechanism to yield an overall quantity
  - Classic PA: counts quantity along a critical path



Pruned BFS traversal of a binary tree (p=3)
Labels indicate assigned (pruned) order

PACO MM & Strassen's Algorithms:

- PACO MM algorithm is then a pruned BFS traversal of a 2-way divide-and-conquer tree
- PACO Strassen is a pruned BFS traversal of a 7-way tree.
- The divide-and-conquer trees are of the sequential divide-and-conquer algorithms.
- If associating the root with p processors and dividing a node with p' processors by the ratio of $\left\lfloor \frac{p'}{2} \right\rfloor : \left\lceil \frac{p'}{2} \right\rceil$ => PACO MM-1-Piece
- If stop the traversal after some constant number of rounds => PACO Strassen-Const-Pieces



Pruned BFS traversal of a binary tree (p=3)
Labels indicate assigned (pruned) order

Extension to Distributed-Memory Settings:

- If each processor has an arbitrarily large local disk, i.e. local VM, a PACO algorithm's communication can be divided into two phases:
  - An inter-processor message passing: the BW will be the memory-independent communication bound
  - A local sequential computation: the BW will be the memory-dependent or memory-independent bound, depending on the relative memory footprint to M
- If assuming a distributed-memory model with only one local memory of size M, but no local disk:
  - BW keeps the same
  - Latency will be about a factor of M lower since it divides the total communication into a sequence of messages of $O(M)$ each

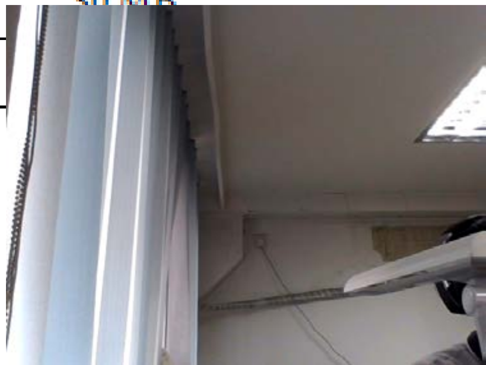
Extension to Heterogeneous Setting:

- A Heterogeneous Model: p processors, each of which can have a different but *fixed* throughput, say FLOPS (Floating Point Operations Per Second)
- A general Heterogeneous PACO algorithm:
  1. Normalize all throughputs (real numbers) to a monotonically non-decreasing order, i.e. $t_1 : t_2 : \cdots : t_p$, and $t_1 = 1, \forall i, j \in [1, p]$, we have $t_i \leq t_j$ if $i \leq j$
  2. Normalize the throughput ratio to fraction ratio of $f_1 : f_2 : \cdots : f_p$, where $f_i = t_i / \sum_{j=1}^{p} t_j$ indicates the fraction of total computational loads for processor-i
  3. Traverse the algorithm's divide-and-conquer tree in a pruned BFS fashion. Each node is associated with its fraction. Root is with 1
     1. In the case of Strassen, a node of size n' has $f' = \left(\frac{n'}{n}\right)^{\omega_0}$, where $\omega_0 = \log_2 7$
  4. Prunes whenever a node's f' is $\leq$ some processor's remaining f_i, then $f_i = f_i - f'$
  5. Repeat the above 3-4 until base cases
- Discussions:
  - Our model is simpler than [BallardDeGe11], which considers four parameters, i.e. β_i (inverse BW), α_i (latency), M_i (local memory size), and γ_i (flops), for $1 \leq i \leq p$
  - Beaumont et al. proposed 2D and 3D approximate Non-Rectangular Partitioning for squared MM on a heterogeneous computing system, with a proof that an exa partitioning is NP-Complete.

Preliminary Experiments on PACO MM-1-Piece:

- All algorithms of the same problem call the same kernel function(s)to compute sequentially base cases => ***compares only the differences in partitioning and scheduling***
- Include all partitioning and scheduling overheads in final running time
- Measure ``running time'' as a min of at least three independent runs
- $speedup = \left(\frac{running\_time_{peer\ alg.}}{running\_time_{PACO}} - 1\right) \times 100\%$
- $Rmax = 2 \times n \times m \times \frac{k}{time\_in\_second}$
- $Rpeak = 24 \times (2.3 \cdot 10^9) \times 16$
  - Each core can perform 16 dual precision flops (by Fused Multiply Add – FMA instr.) per cycle

Name	24-core machine
OS	CentOS 7 x86_64
Compiler	ICC 19.0.3
CPU	Intel Xeon E5-2670 v3
Clock Freq	2.30 GHz
# sockets	2
# cores / socket	12
Dual Precision FLOPs / cycle	16
L1 dcache / core	32 KB
L2 cache / core	256 KB
L3 cache (shared)	30 MB
memory	



INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

Preliminary Experiments on PACO MM-1-Piece:

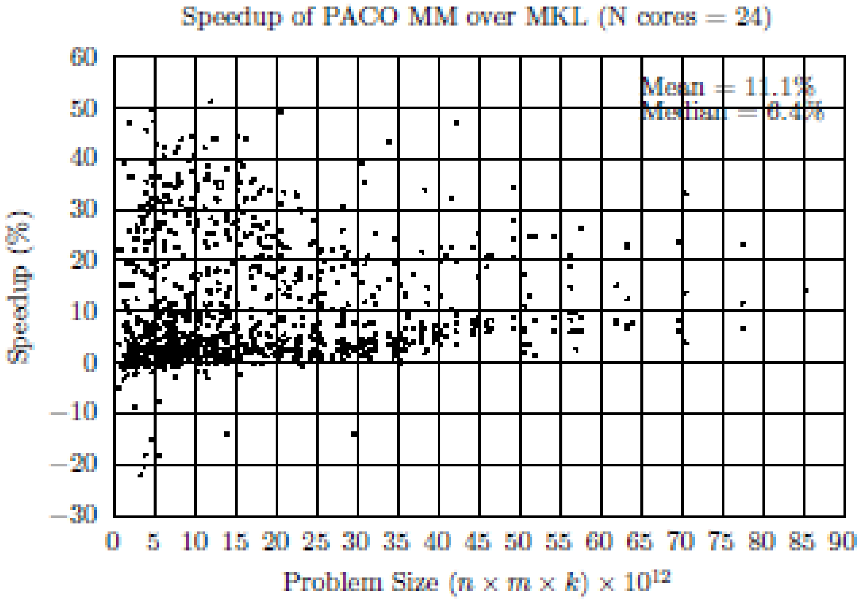


Figure 2: PACO MM-1-PIECE algorithm's speedup over Intel MKL's dgemm

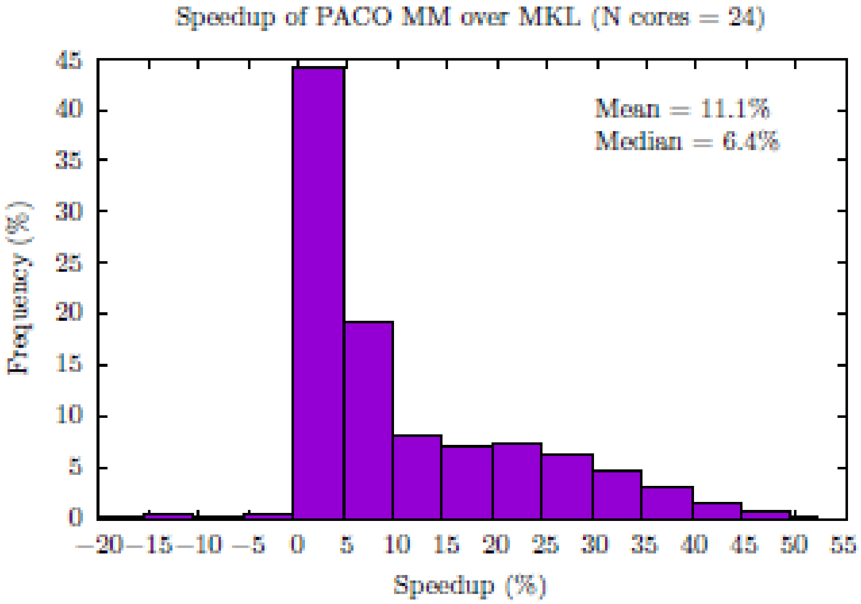


Figure 3: Distribution of the speedups

Rmax/*Rpeak*	PACO	MKL	CO2
Mean	82.6%	75.1%	37.8%
Median	84.0%	78.4%	39.3%

Table 3: *Rmax*/*Rpeak* of MM algorithms. "CO2" is the PO depth-*n* MM algorithm based on 2-way divide-and-conquer with a base-case size of 64 [36].

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

Preliminary Experiments on PACO MM-1-Piece:

- Problem size is calculated as $n \times m \times k$, where n, m, k iterate independently from $8{,}000$ to $44{,}000$ with a step size of $4{,}000$
- 24-core machine:
  - The mean and median speedup of PACO MM-1-Piece algorithm over Intel's MKL is 11.1% and 6.4%, respectively
  - A PO algorithm's implementation requires to tune a proper base-case size:
    - Recent research by Leiserson et al. [LeisersonThEm20] shows that a well-tuned PO MM algorithm achieves about 40% of machine's peak performance
    - If a base-case size too small, it increases the ``slackness'' of algorithm and allows better processor utilization for a wider range of processor counts, but at the cost of more deviations from its sequential execution order, hence more communication and synchronization overheads.
    - If a base-case size too large, a base-case task may not fit in some upper-level cache(s) of processor, hence it may not be cache-efficient, and the load imbalance an processors may be larger, i.e. some processors may be under-utilized

  ✓ ***Our approach does not need to tune***.

Concluding Remarks:

- A general PACO algorithm: partition comp. and comm. Evenly and recursively among p processors by a ***pruned BFS*** traversal of a cache-oblivious algorithm's divide-and-conquer tree.
- Application to several important cache-oblivious algorithms, including general MM and Strassen
- Compared to classic PA: our algorithms achieve perfect strong scaling on an arbitrary number, even a prime number, of processors within a certain range. The comp. and comm. Imbalance among processors, if any, is an asymptotically smaller term, rather than a larger-than-1 multiplicative factor.
- Compared to classic PO: our algorithms usually have better comm. Complexities.
- Our PACO Strassen-Const-Pieces algorithm provides an almost exact solution to the ***Open Problem*** on parallelizing Strassen's algorithm ***efficiently*** and ***exactly*** on an ***arbitrary*** number of processors.
- Extensions to distributed-memory and heterogeneous settings.
- Provides a new perspective on the fundamental ***Open Problem*** of extending t recursive cache-oblivious technique to an arbitrary architecture.

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

Q &A ?