Experiments with List Ranking for : Explicit Multi-Threaded (XMT) Instruction Parallelism

S. Dascal and U. Vishkin

Fascinating motivation:

Will a 2cm $\times$ 2cm Piece Of Silicon Provide PRAM-on-chip Breakthrough?

See http://www.umiacs.umd.edu/~vishkin/XMT/.

General Objective

Technical goal Use on-chip resources to reduce single task completion time for "general-purpose" computing.
Opportunity Next decade: "Billion transistor chip era"; $> 10^4$ increase from 1980s.
Question **"What to do with all this hardware?"**

Current uniprocessing: Limited parallelism.
Current multiprocessing: Coarse-grained SW&HW parallelism due to high overhead SW&HW mechanisms; hence programmability problems. Large inputs.
On-chip: Diminishing returns for adding device; bandwidth no problem (Dally-Lacy'99). **Need: new context & low overhead (e.g., HW) mechanisms.**

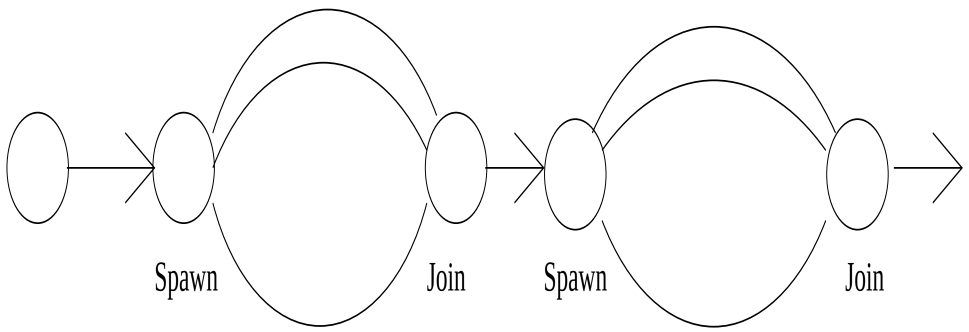
Background: PRAM (parallel random-access model) algorithmics: *knowledge-base 2nd only to serial algorithms.*
General idea: **PRAM-on-chip?**
build on PRAM alg & MPP ideas (e.g., Ultra,NESL,SB-PRAM); seek SW&HW mechanisms for more instruction-level parallelism (ILP).
For code compatibility (without more ILP): "standard serial code" = single PRAM processor.

Spectrum of Explicit Multi-Threading (XMT) Framework

Algorithms $--->$ architecture $--->$ implementation.

- XMT: design point for fine-grained parallelism
- New elements are added only where needed

Snapshot: XMT High-level and assembly languages



XMT: from Alg to Prog. to Arch

XMT algorithms Start with Arbitrary CRCW (concurrent read concurrent write) PRAM algorithms.
(Note: Generally sufficient!)

XMT-C: High-level Language An arbitrary CRCW SPMD (single program multiple data) extension of standard C. Includes Spawn and Prefix-sum (PS) statements. Note: Using C-style scoping, Joins are implicit.
Spawn creates threads; each progresses at its own speed; expires at its Join. (New feature: "Independence of order semantics (IOS)".) Synchronization: only at the Joins. Threads avoid busy-waits by expiring.

Multi-operand instruction *Prefix-sum* (PS) instruction , `PS Ri Rj`, has an "atomic" outcome:
(i) Store `Ri + Rj` in `Ri`, and
(ii) store original value of `Ri` in `Rj`.
Note: Fetch&Add semantics. Several PS instructions may be combined at run-time to form one *multiple-PS* instruction. A new multi-operand PS functional unit executes the PS instruction.

PRAM algorithm $-->$ XMT-C automatic+programmer (semi-auto); longer threads (for more locality and less synchrony); PS implements Arbitrary CW.

XMT-ASL "instr set arch" Standard assembly language, plus 3 new instructions: Spawn, Join, and PS.

XMT-C $-->$ XMT-ASL Compiler.
Execution model for XMT-ASL Constructive version of the elegant trace-based execution models for studies of ILP in Austin-Sohi, Lam-Wilson, Wall. Accounts for intra-thread parallelism, cache misses, prefetching, etc.

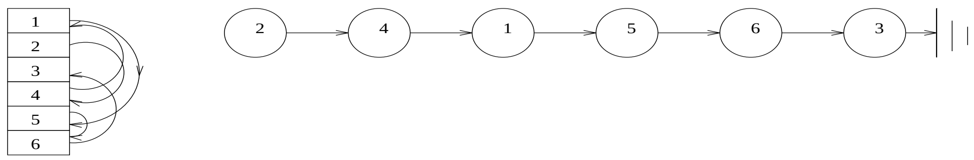
XMT "HW architecture" Thread control units (TCUs); local registers; (degenerate) PS function unit; HW impl of Spawn and Join. (Thread Allocation. Given: Spawn (of n virtual threads). Once a TCU completes one thread, quickly reuse it to execute another thread.)

XMT-ASL $-->$ XMT HW-arch Scheduling Lemma.

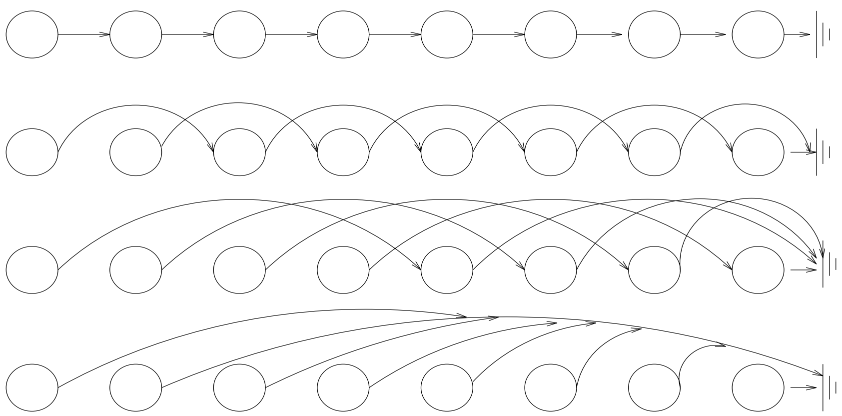
But, why this involved XMT-ASL + execution model? This is the way in which it will be implemented.
"Present things as simply as possible, but not simpler than that" (A. Einstein?).

List Ranking

Problem definition For each element, find distance from end of list.



Serial algorithm. Trivial.
Basic parallel list ranking (Wyllie). $W = O(n \log n)$, $T = O(\log n)$.



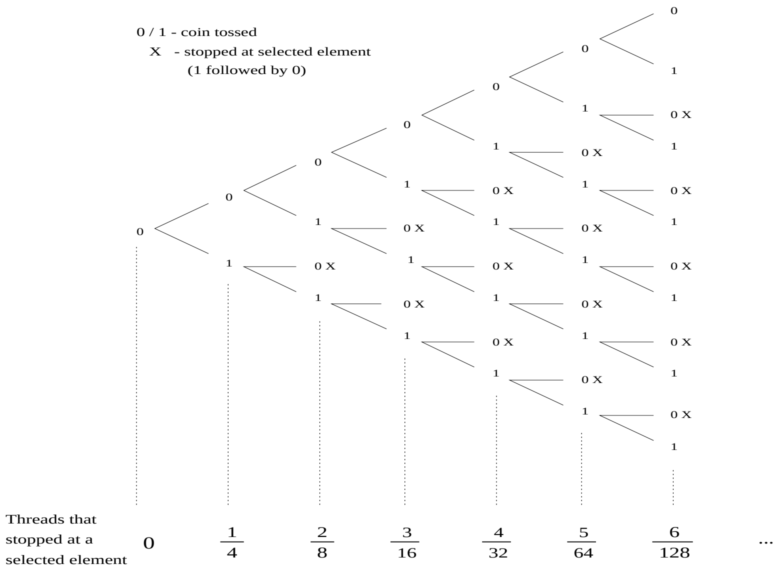
Fundamental symmetry breaking: theory

Deterministic. "Deterministic coin tossing". Traff: for small p speedups not satisfactory.
Randomized by coin tossing, V-84. Toss coins. Pick H followed by Tail. Shortcut. Iterate.
Randomized by random selection, Reid-Miller-94. Pick s random elements each between 1 and n (for some s). Eliminate multiplicities. Shortcut from one picked element to the next. Iterate.

Algorithms studied

Serial
Wyllie
No-cut Toss coins. Pick H followed by Tail. Shortcut to the next picked. Iterate.
Cut-6 Toss coins. Pick H followed by Tail. Shortcut 6 or to the next picked, whichever comes first. Iterate.
RM for comparison only; so RM results are favorably biased.

Analytic/empiric notes



No-cut Expected length of longest chain $E(Maxlen)$ is $1.2 * \log n$. (Since $i --> i+1$: $\#$ length $> i$ chains is nearly 1/2.)

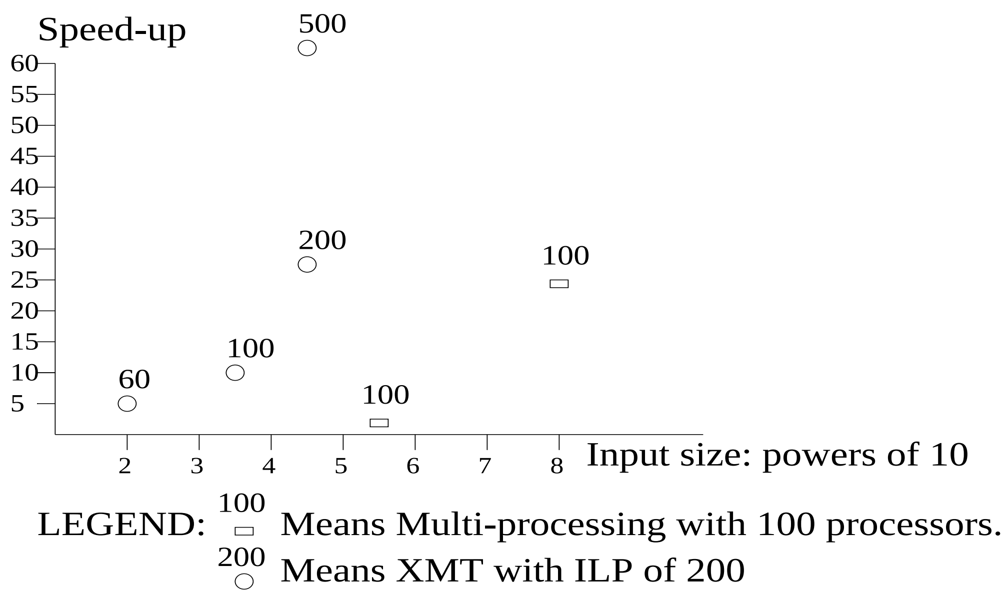
Cut-6 Among Cut-i least time and nearly least work. Time $O(\log n)$; not $O(\log^2 n)$ as for No-cut and RM.

RM
- Less work: only a picked element spawns a thread.
- $E(Maxlen)$ is $2.75 * \log n$; with 1/4 selected $i --> i+1$: $\#$ length $> i$ chains is $>$ 3/4. - For smaller inputs T dominates $W/P\ +\ T$.

Empirical Results

Serial Least work; most time.
Wyllie Most work; least time.
No-cut Among parallel: least work for smaller inputs.
Cut-6 More work and less time than no-cut.
RM Among parallel: least work for larger inputs; most asymptotic time.

Some XMT speedups (relative to best serial); $P = 1000$:
For $n = 4.2 * 10^6$, RM: 270, and No-Cut: 154.
For $n = 8 * 10^3$, RM: 34, and No-Cut: 51.



• MPP versus XMT. List ranking results for the Paragon reported in Sibeyn-97 versus our (simulation) XMT results. One reason: dynamic load balancing in XMT.

Notes on Experimental Evaluation

- Execution model: reflects intra-thread parallelism, cache misses, prefetching.
- Compiler feasibility.
- A conservative comparison approach.

Avoid unfairly favoring the performance estimates of the XMT code over standard serial code. Concretely, (1) For standard serial code, we took its (parallel) T as the running time. Rationale: this is a **lower bound** on the (parallel) execution time achievable.
(2) For parallel code, we took its $W/p + T$ as the running time. Rationale: this is an **upper bound** on the execution time achievable.

Grains of Salt

No-cut: optimized to compete with Cut-6. Could have looked better relative to RM.

RM: Multiplicities eliminated by a kind of PS; discount...

Experimental Algorithms/List Ranking Conclusion

• Example of incorporating analytic (non-asymptotic) performance analysis into experimental performance analysis (for small inputs).

• Good speedups for small inputs are possible.

• Why care for speed-ups for smaller inputs?
Hypothetical numeric example S - base serial system. Compare A&B, two parallel systems. Suppose: For 95% of the running time on S, system A runs 100 times faster; for 5%: running time unchanged. System B runs well also on smaller inputs. For 99% of the running time on S, system B runs 100 times faster; for 1%: running time unchanged.
How systems A&B compare? Amdahl's Law: Factor ≐3!

• Demonstrated interplay between:
Computing system thinking and Algorithm engineering
Add to WAE scope?

XMT Conclusion

• Fundamental differences w.r.t. Cray-vector and VLIW: for performance, they couple activities not necessitated by program semantics; but, delay in one delays more. Such "tightly-coupled orchestration" may not fit technological constraints: the inability to predict response time to memory requests.

• Using multiple program counters (Multiscalar).
SMT: same cycle execution of instructions from different programs.
None aspires/claims to support PRAM-like algorithms.

• **M$?: PRAM alg implementable?**
Not a minute too early to try (even if theorist...).
New ground rules; rich MPP&serial idea base.

• Need to exploit parallelism: due to fundamental limits; not just technological predictions as when funding focused on parallel computing! Expected: <16% of a chip accessible in a clock cycle! Also, miniaturization reduces speed!

• Quest for "paradigm shift" (T. Kuhn) in CS&E
Find a general-purpose way for minimizing computation

completion time by way of parallelism.