Efficiently Speeding up Sequential Computations with N-way parallelism

Romain Cledat, Tushar Kumar and Santosh Pande

College of Computing, Georgia Institute of Technology

Motivations

Utilizing parallel resources

Single-core Resource

- Frequency scaling
- No effort to use "more" (faster)

Many-core Resource

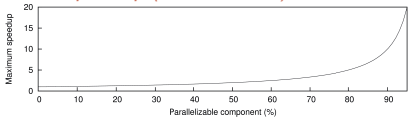
- Difficult to program
- Task/data parallelism not always applicable

Resources are wasted

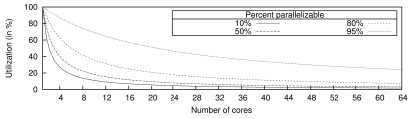
+

Sequential bottleneck

Limits speedup (Amdahl's law)



Limits utilization



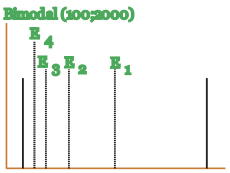
=>

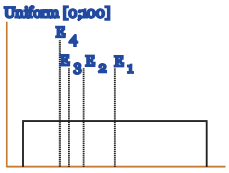
- How to occupy parallel resources?
- How to do it efficiently?
- How to overcome the sequential bottleneck?

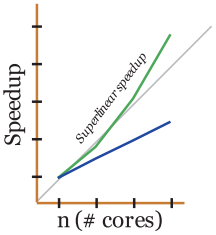
Think Different: How can parallel resources be used to improve sequential performance?

Intuition: Randomized algorithms

- An algorithm making random choices for a fixed input can lead to different completion times







Idea: Launch multiple instances

- Launch n parallel independent and isolated instances of the kernel algorithm on the same input
- Fastest among n is faster than average with high probability
- Use of parallel resources to speed-up hitherto sequential kernels
- Potential for super-linear speedup for certain distributions
- Applicable to other algorithms using diversity

Efficient parallelism

- More instances => more speedup
- Is there an "efficient" number to launch?
- Two approaches: learning or culling

Definition of efficiency

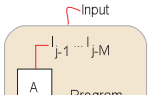
- S_n: Speedup obtained with n instances
- e: Parallel efficiency defined as $\frac{S_n}{n}$
- $e = 1$ is a linear speedup and $e > 1$ is super-linear

Learning approach for randomized algorithms

- Monitor past executions and predict E_1 to E_n

$$CDF_n(t) = 1 - (1 - CDF_1(t))^n$$

- Assume stable behavior
- Calculate n based on computed efficiency



Stability Condition

$$PDF[A(I_j)] \approx PDF[A(I_{j-1}) \ldots A(I_{j-M})]$$

- Holds **statically** for inputs of the same "size"
- Holds for **sufficiently slow variations**

Culling approach

- Launch as many as possible
- Monitor behavior and cull worst performing
- Indirectly affects efficiency

Status Monitor

- Domain-specific progress report
- Triggers culling

```
Input: Way.t ways[] to evaluate for culling
Input: int n number of ways
Output: Way.t toCull[] to cull
Data: Way.t classes[][]
classes[0] ← {ways[0] ... ways[n]};
newClass ← 1;
lastClass ← 0;
while newClass do
    newClass ← 0;
    foreach way w1 in classes[lastClass] do
        foreach way w2 in classes[lastClass]
        ordered after w1 do
            if not isSimilar(w1, w2) then
                if newClass == 0 then
                    lastClass ← lastClass + 1;
                    newClass ← 1;
                end
                append(classes[lastClass], w2);
            end
        end
    end
end
worstClass ← pickworst(classes); /* Empty if only one class */
```

Extension for heuristics

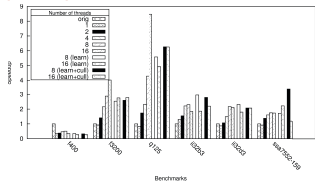
- Monitor past executions and predict $E[H_1]$ to $E[H_n]$
- Estimate execution time of non-completing heuristics
  - Non-launched H_i estimated at $d.E[H_i]$ (d<1)
  - Launched estimated at $c.E[H_w]$ (c>1)
- Greedily pick fastest heuristics and compute efficiency

Conclusion

- N-way is *easy*
- Determining n is hard
  - Too small n does not provide all potential speedup
  - Too big n wastes resources and can harm speedup
- Learning approach estimates best n while culling further limits waste

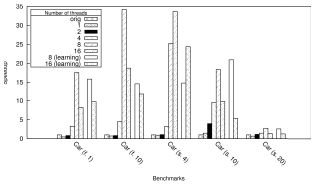
Results: Speedups and QoR improvements with low overhead

Speedup Results



WalkSAT benchmark

WalkSAT solves SAT problems using randomness. The benchmarks used are from the DIMACS suite



Motion Planning (MSL) benchmark

MSL implements rapidly-exploring random trees. The Car benchmark is a path planning hand-crafted benchmark

QoR improvements

Threads	krob150	a280	lin318	rat575
Original	44.7	7.8	139	31.5
1	44.8	7.8	139	31.5
2	44.8	7	134.3	29.3
4	41.6	6.8	128.2	28.7
8	39.1	6.5	122.3	27.7
16	38.4	6.3	113.8	26.7
8 (l)	42.4 (4.9)	7.1 (4.6)	126.9 (4.5)	28.7 (4.1)
8 (l+c)	45.1 (3.5)	7.1 (3.5)	132.5 (3.9)	29.1 (3.8)
16 (l)	40.1 (8.3)	7 (7.0)	128.9 (7.3)	28.9 (6.6)
16 (l+c)	43.7 (8.1)	7.2 (7.0)	132.3 (7.2)	29.3 (6.6)

TSP benchmark

"Fitness" of the population after 2500 generations for benchmarks from TSPLIB (smaller is better)

Overhead Measurements

Function	% total overhead			Call time (ns)		
	1	8	16	1	8	16
Access (RO)	53.78	48.49	49.72	12	13	26
Access (RW)	24.72	31.69	24.11	9	13	21
Thread info	14.95	13.92	18.53	2	2	5
Thread private	5.85	4.45	6.15	24	24	57

Runtime Overhead

Principal overhead for the n-way runtime for 1, 8 and 16 threads. "Thread info" represents access to the thread's characteristics and "thread private" corresponds to a fetching of TLS information. All other runtime functions represented less than 1% of total overhead.

Current and future work

Evaluating algorithmic diversity

- Define and quantify *diversity* between algorithms (data access patterns, timing information, ...)
- Derive a *fingerprint* for algorithms
- Dynamically learn fingerprints for each way to maximize diversity expressed

Exploiting hardware diversity

- Current work exploits *algorithmic diversity* but hardware also exposes diversity
- Heterogeneous chips, accelerator-based systems are all examples
- N-way can be used to dynamically match the algorithm, the data and the hardware

Traditional parallelism versus N-way

- Orthogonal approaches that can be used simultaneously

Parallelism as a source of diversity

- Many ways to parallelize an algorithm, introducing diversity in parallelization
- Simple examples: grain-size, algorithm break-up boundaries, ...
- N-way can be used to simultaneously evaluate different parallelization options
- Determining correct 'n' becomes even more crucial

Side by side

- Evaluate whether diversity or traditional break-up provides better speedup
- Determine best scheduling and balance between n-way and parallelism
  - Current learning approaches estimate N-way's efficiency
  - Efficiency can also be learned/estimated for traditional parallelism

References

- R. Cledat, T. Kumar, J. Sreeram, S. Pande: *Opportunistic Computing: A New Paradigm for Scalable Realism on Many-Cores*, HotPar 2009
- R. Cledat, S. Pande: *Energy Efficiency via the N-Way model*, PESPMA 2010