Opportunistic Computing: A New Paradigm for Scalable Realism on Many-cores

Romain Cledat and Tushar Kumar

Georgia Institute of Technology

Advised by Professor Santosh Pande

Using Computing Resources

Single-core Resource

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

Multi-core Resource

- Task and data parallelism
- Hard but still tractable

Many-core Resource

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

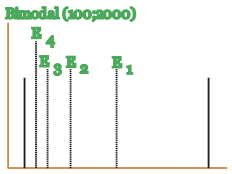
Resources are wasted

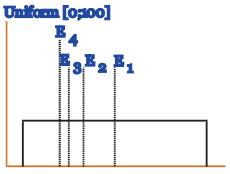
Think Different: How can we usefully utilize resources differently and what for

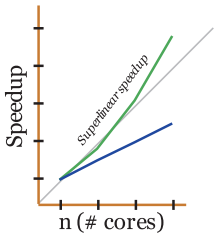
N-version: Speeding up certain sequential computation

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

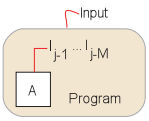
- S_n: Speedup obtained with n instances
- e: Parallel efficiency defined as $\frac{S_n}{n}$
- $e = 1$ is a linear speedup and $e > 1$ is super-linear

Learning approach

- 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

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

- When does it hold?
- Runtime estimation required

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

Culling approach

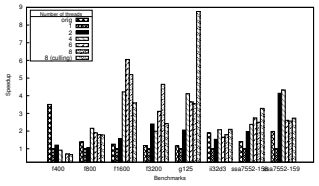
- Launch as many as possible
- Monitor behavior and cull worst performing
- Indirectly affects efficiency

Status Monitor

- Domain-specific progress report
- Triggers culling

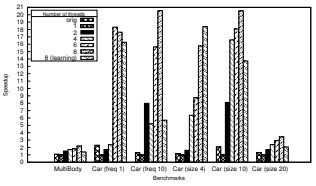
```
Input: Way t ways[] to evaluate for culling
Input: int n number of ways
Output: Way t toCull[] to cull
Data: Way t classes[][]
classes[0] ← {ways[0], ..., ways[n]};
newClass ← 1;
lastClass ← 0;
while newClass do
    newClass ← 0;
    foreach way w1 in classes[lastClass] do
        foreach way w2 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 */
```

Results: Low-overhead runtime provides good speedup



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

SRT: Scaling semantics to available resources

Intuition: Real-time systems

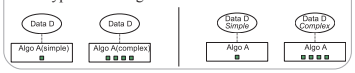
- Algorithms whose execution time, multi-core resource requirements and sophistication are parametric can be scaled to maximize sophistication within **responsiveness** constraints

What about RT methods?

- Requires applications to be decomposed into tasks and precedence constraints. Responsiveness is guaranteed by RT semantics (hard or probabilistic)
- For games and multimedia:
  - · Implemented with high-productivity programming flows (C/C++ with 100K+ LOC)
  - · No analyzable responsiveness structure
  - · Responsiveness is an emergent attribute (tuning is an art)

Algorithm scaling

- Provides additional mechanism to utilizing more cores
- Two types of scaling: with resources and with data

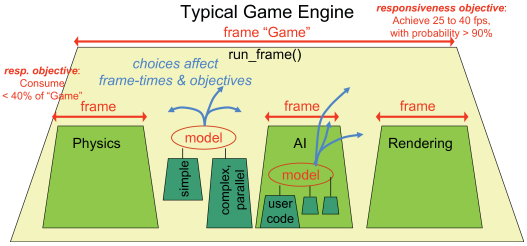


Semantics scaling for monolithic applications requires a new bag of tricks

SRT system architecture

The SRT Runtime:

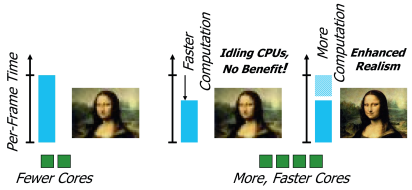
- Monitors the frames
- Learns application-wide average frame structure
- Chooses between user-codes in model



Reinforcement learning: infers complex model-objective relationships (slow)

Feedback control: adjusts model choices to meet objectives (fast reaction)

Enabling Realism



N-version creates slack by speeding up computations

SRT exploits it by achieving more within the ressource constraints

Realism...

- **Sophistication in modeling**
  - · *Example*: Render/animate as detailed as possible
- **Responsiveness**
  - · *Example*: Frames-per-second, user-input response time

Enables richer applications

- More immersiveness in games
- More adaptive media processing

Scope

- Each framework can apply separately but more powerful together
- Gaming, multimedia applications
- Immersive applications

References

- R. Cledat, T. Kumar, J. Sreeram, S. Pande: *Opportunistic Computing: A New Paradigm for Scalable Realism on Many-Cores, HotPar 2009*