Vector Load Balancing in Charm++

Ronak Buch
Parallel Programming Laboratory, University of Illinois at Urbana-Champaign

October 21, 2020
18th Annual Workshop on Charm++ and Its Applications
Load Balancing

- Load balancing is a hallmark of Charm++
- Performance often limited by maximum load on a PE
- RTS measures load and migrates objects in response
- Dynamic, irregular applications have been able to achieve high performance and scalability because of it
What is Load?

- *Load* is really just a proxy value we use to reason about performance
  - In truth, we want to minimize execution time
  - Unbalanced, fast program > balanced, slow program

- CPU time per object by itself is often a sufficient metric for this value

- However, in the same way measuring cache misses or pipeline stalls improves upon merely profiling, sometimes more detail is helpful
Vector Load Balancing

• Rather than being a single value, *load* is now a vector of multiple values
  ◦ Store vector loads in LBDatabase
  ◦ Pass vector loads to strategies
  ◦ Use vector loads in strategies

• Can be used generically: for various hardware measurements (CPU/GPU/network/memory), discrete parts of an iteration, application specific parameters, etc.
Vector Strategies

- Extra dimensionality makes vector load balancing computationally difficult
- Objects can no longer be totally ordered
- Want to minimize the maximum in each dimension
- NP-complete problem, so only interested in approximations
Vector Strategies

• A simple strategy finds object with global maximum load dimension and places it on PE with minimum load in that dimension
  ○ Only works well when object has load in only one dimension

• For more realistic cases, have to consider vector holistically
Vector Strategies

- Find object with maximum $p$-norm and place on PE with minimum $p$-norm after placement
  - Works well, but computationally expensive
  - PE “weight” varies with object, i.e. $\| (2, 0) \|_2 < \| (0, 3) \|_2$, but when adding $(3, 0)$, $\| (5, 0) \|_2 > \| (3, 3) \|_2$

- Calculate average load vector in $d$-space and create a normal hyperplane, then repeatedly allow furthest PE below the hyperplane to choose an object
New Load Balancing APIs - Phase

- Many applications have orthogonal *phases* within an iteration separated by barriers (or weaker synchronization)
- New functions have been added to track phases for load balancing:
  - `void CkMigratable::CkLBSetPhase(int phase)` - Until called again, all automatic LB measurements for calling chare attributed to specified phase
  - `int CkMigratable::CkLBGetPhase()` - Returns current phase
New Load Balancing APIs - Manual

- Added new API for recording vector load data
  - `void CkMigratable::CkLBSetObjTime(LBRealType load, int dimension)` - Sets specified dimension of vector load for calling chare
  - `std::vector<LBRealType> CkMigratable::CkLBGetObjVectorLoad()` - Returns current vector load for calling chare
Using Vector Strategies

- Currently only strategies built on top of TreeLB support vector load balancing
  - TreeLB is new flexible, optimized replacement of CentralLB and HybridLB
  - Eventually all non-distributed strategies should use TreeLB
- If vector loads are detected in the LB database, a vector version of the chosen strategy is automatically used if available
Writing Vector Strategies

- Objects and PEs are templated on dimension, replicated in a `static constexpr` field for external access.
- A specific dimension of Object or PE load is accessible with `LBRealType getLoad(int dimension)`.
- Template specialization allows LB author to handle vector and non-vector cases.
template <typename O, typename P, typename S>
class Example : public Strategy<O, P, S> {
    public:
        void solve(std::vector<O>& objs, std::vector<P>& procs,
                    S& solution, bool objsSorted) {
            // vector implementation
        }
};

template <typename P, typename S>
class Example<Obj<1>, P, S> : public Strategy<Obj<1>, P, S> {
    public:
        void solve(std::vector<Obj<1>>& objs, std::vector<P>& procs,
                    S& solution, bool objsSorted) {
            // scalar implementation
        }
};
Vector LB Performance - AMPI

AMI - No Load Balancing

Time In Microseconds

0 100,000,000 200,000,000 300,000,000

PE 0
(57, 57)

PE 1
(57, 57)

PE 2
(57, 57)

PE 3
(57, 57)

AMI - No Load Balancing
Vector LB Performance - AMPI

AMI - Regular Load Balancing

Time in Microseconds

0 100,000,000 200,000,000 300,000,000

PE 0
(73, 57)

PE 1
(73, 57)

PE 2
(73, 57)

PE 3
(73, 57)

AMI - Regular Load Balancing
Vector LB Performance - AMPI

Time in Microseconds

0 100,000,000 200,000,000 300,000,000

PE 0
(77, 57)

PE 1
(77, 57)

PE 2
(77, 57)

PE 3
(77, 57)

AMPI - Vector Load Balancing
Vector LB Performance - AMPI

LB Off

Phase Unaware
(1.44x speedup)

Phase Aware
(1.67x speedup)
Vector LB Performance

Timeline of phase-based application:

[Diagram showing a timeline with phases for PE 0, PE 1, PE 2, and PE 3, each with specific time intervals marked.]
Vector LB Performance

No LB
Vector LB Performance

(non-vector) GreedyLB
Vector LB Performance

Vector Greedy
Applications

- **ChaNGa**
  - Working, but no performance results at scale yet
  - Time spent in each rung of multi-stepping corresponds to dimension in vector

- **NAMD**
  - In process of making vector of CPU and GPU load

- Please contact me if you think your application would benefit!

Ronak Buch  rabuch2@illinois.edu
Future Vector LB Work

• Performance is still an issue, so optimizations needed
  ○ Discretization, clustering, space-partitioning, etc. should go a long way

• Exploit distribution of load per-dimension

• Integrate HAPI into load measurement to automatically record accelerator load

• Add support for constraint based objective functions for cache/memory balancing
Conclusions

- Applications often have scope for improved load balance
- As programming techniques and hardware become more complex, this scope will likely increase
- Providing more detailed load data via Vector LB has been shown to improve decision quality over traditional LB in testing