Load Balancing of Irregular Parallel Divide-and-Conquer Algorithms in Group-SPMD Programming Environments

Mattias Eriksson Christoph Kessler Mikhail Chalabine

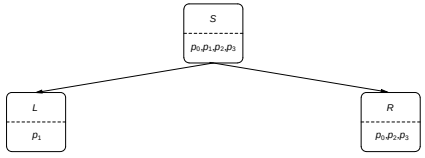
Linköping university, Sweden

Originally presented at the PASA workshop — 16 mars 2006

Outline

Parallel Quicksort

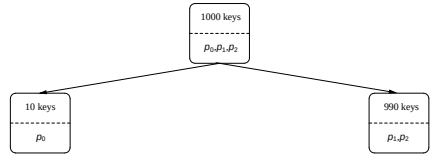
- A processor group selects a shared pivot and partitions S into $L = \{x \in S | x \leq \pi\}$ and $R = \{x \in S | x \geq \pi\}$.
- The processor group splits into two groups.
- The first processor group sorts L and the second processor group sorts R.



- Examples of DC-algorithms with data-dependent (irregular) subproblem sizes: quicksort and quickhull.
- Examples of DC-algorithms where subproblem sizes are fixed: mergesort, FFT and Strassen matrix multiplication.

Imbalanced workload in parallel quicksort

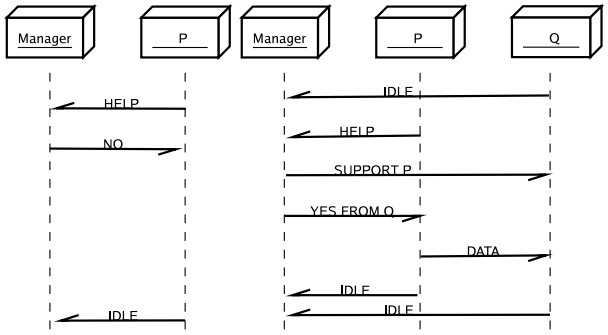
With subproblem sizes being data-dependent a good processor assignment to subproblems is not always possible.



- Imbalance may accumulate in the parallel phase.
- Computing resources are wasted.

The manager strategy

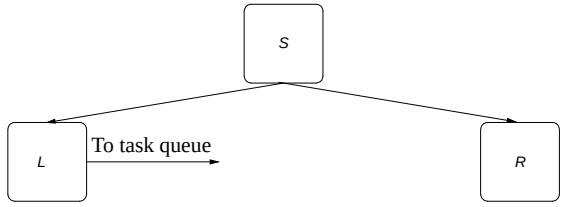
- One of the processors is dedicated to coordinating the others.



+ A processor with high workload can get help from an idle processor.
- The manager does no sorting at all.

The task queue strategy

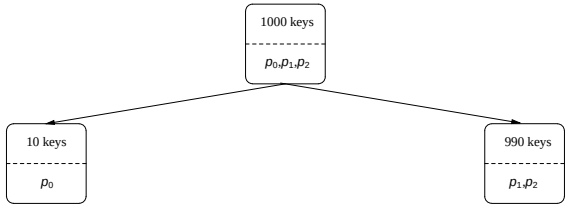
- After partitioning is done in the one-processor phase, one of the subtasks is put in a shared task queue.
- When a processor has no more work to do, it fetches a new task from the queue.



+ The load imbalance will be very small.
- Managing the task queue will cause overhead.

The repivoting strategy

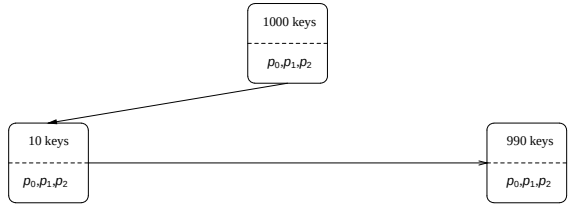
- If processor groups will become imbalanced a new pivot is selected.



+ Bad load balance between processor groups is avoided.
- When a new pivot is selected the partitioning will have to be redone.
- Does not work where pivot is uniquely determined (eg. quickhull).
- Exactly how do we define bad load balance?

The serialisation strategy

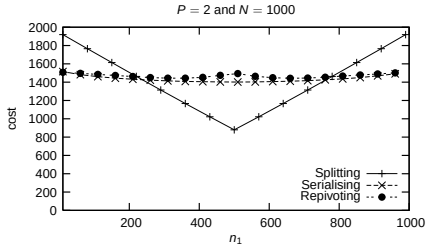
- If processor groups will become imbalanced the whole processor group will first sort the first subproblem and then the other subproblem.



+ No load imbalance at all when serialising the execution in the parallel phase.
- The processors will execute more work in the parallel phase.
- Exactly how do we define bad load balance?

Optimal local strategy: example

Question: When is it worthwhile to serialise/repivot?
Example: 2 processors sort 1000 elements. n_1 is size of the first subproblem.

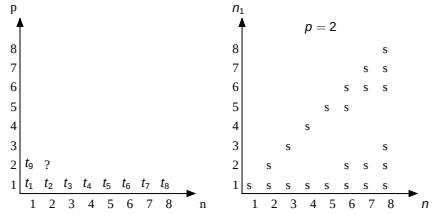


- Optimal strategy is sometimes repivoting/serialisation and sometimes group splitting.
- We want to find the threshold values!

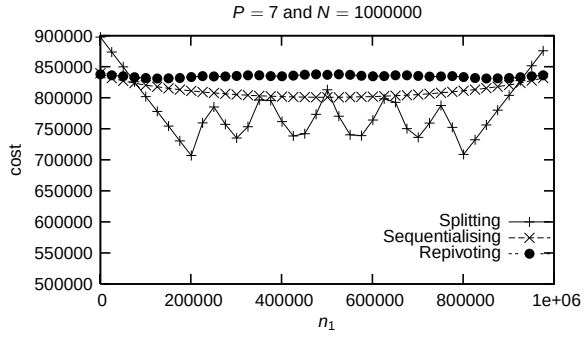
Optimal local strategy: numerical approach

Question: When is it worthwhile to serialise/repivot?
Find out numerically with dynamic programming

- Dynamic programming is a method for solving problems of a recursive nature.
- Calculate $\mathrm{E}[T_p(n)]$, for increasingly large p and n, using previously calculated values.
- Also calculate which strategy of group splitting, repivoting and serialising is best for a given situation.



Having a look at the results



- $n_1 = n/2$ is not the optimal split point!

Results of dynamic programming

We found from dynamic programming that:

- On average serialisation is always better than repivoting.
- Serialisation is better than group splitting when one of the subproblems is very small compared to the other.

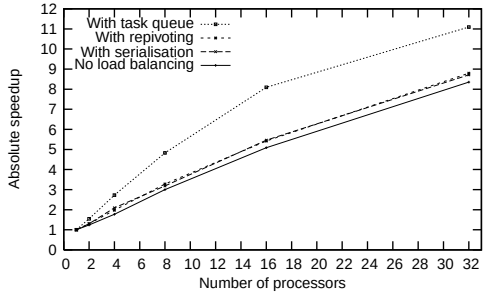
MPI-thresholds

p / n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10^5	.156	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10^6	.328	.216	.136	.076	.022	0	0	0	0	0	0	0	0	0	0
10^7	.382	.265	.191	.143	.109	.083	.062	.042	.026	.011	0	0	0	0	0
10^8	.401	.281	.206	.159	.126	.102	.083	.067	.054	.043	.033	.025	.015	.008	0
10^9	.407	.284	.210	.163	.131	.107	.088	.073	.061	.051	.042	.034	.027	.021	.016

- Example: Consider the case $p = 3$ and $n = 10^7$. If $n_1/n < .265$ serialisation is better than group splitting.
- We implement a hybrid local heuristic, parameterized by this table (or a corresponding table for Fork).

Execution times in Fork

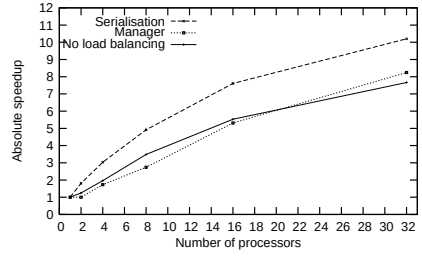
Speedups of quicksort running on the SB-PRAM simulator.
In this example 40000 integers are sorted.



- The dynamic load balancing strategy with a task queue performs better than our strategy with serialisation.
- SP-PRAM: Nonblocking task queue operations, low overhead.

Execution times in MPI

Speedups of quicksort on a cluster with Intel Xeon processors.
In this example 100 million integers are sorted.

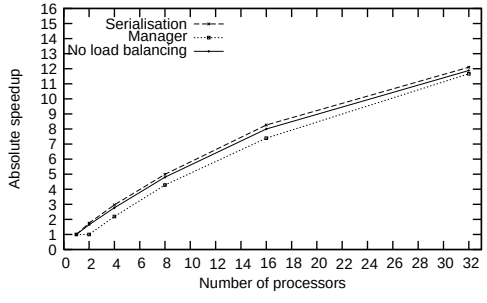


- Here our local load balancing strategy with serialisation outperforms the dynamic manager solution!
- One processor dedicated; communication overhead.

Execution times in MPI with a better pivot

- Early investing more time in better pivoting can reduce need for load balancing.
- Example:

Speedups of quicksort in MPI.
In this example 100 million integers are sorted.
Pivot is selected as average of 7 elements.



Conclusions

- We have proposed and studied new strategies for local load balancing in SPMD: repivoting and serialisation.
- We developed an optimal hybrid local load balancing method, calibrated with threshold tables derived from offline dynamic programming.
- Our hybrid local strategy outperforms the global dynamic load balancing (manager solution) on the MPI cluster.
- Global dynamic load balancing (task queue solution) outperforms our local hybrid strategy on the SB-PRAM where synchronization is very cheap.