

Dynamic Load Balancing of Networked Multi-core Computers

Stanley Chien, Gun Makinabakan,
Akin Ecer, and Hasan U. Akay

Computational Fluid Dynamics Laboratory
Purdue School of Engineering and Technology
Indiana University-Purdue University Indianapolis (IUPUI)
723 W. Michigan Street, Indianapolis, IN 46202, USA
E-mail: schien@iupui.edu

Background – Multi-core Computers

- Continuous increasing of computers' system clock speed is reaching its physical limits
- The high clock speed create heat problem for packaging
- The trend of new processors are multi-core systems.
- Most current multi-core processors have multiple identical CPUs

Background

- Computers used in an institution

- Most new PCs and workstations are using multi-core processors
- Number of CPU cores varies with different kind of computers
- An institution will have network of many multi-core computers of different computation capabilities
- Using the available computation power of multi core computers to do parallel computation

Problems to be solved

- Dynamic load balancing of parallel jobs on networked multi-core computers
 - We have developed dynamic load balancing tool for Unix/Linux and Windows based computers for single core processors in the past
 - We augmented the dynamic load balancing tool to make it support multi-core computers

Approach for developing the DLB tool for networked multi-core computers

Approach steps to solve the problem:

1. Find the relative computation speed of each multi-core computer
2. Find the extraneous workload on each computer
3. Find the workload of every process of a parallel job.
4. Use a cost function to predict the effective computation time of each parallel process under a given process distribution
5. Find a load distribution that minimizes the elapsed program execution time.

Approach step 1

- Find the relative computation speed of each multi-core computer
- Measure the execution CPU time of one small benchmark program at each computer's system start up time (CPU time of one CPU core)
- The relative speed of one CPU core of a computer respect to one CPU core of another computers is the ratio of the measured execution CPU times of corresponding computers

Approach step 2

- Find the extraneous workload on each computer of
 - Use ps command in UNIX or Microsoft provided free software, *PsTools*, to measure periodically the percentage of CPU time used by each process running on every computer
 - Identify which load are parallel processes and which load are single processes.
 - Counts periodically the number of single processes that run over 5% of one CPU core time
 - The processes run under 5% of one CPU core time are usually system demons and other negligible processes

Approach step 3

- Find the workload of processes of a parallel job

- Include a Profiling Library in the application code
- During the execution of the application job, the library periodically provides
 - elapsed execution time,
 - elapsed communication time,
 - CPU time, and
 - communication topology information

of each parallel process

Approach step 4

- Use a cost function to predict the effective computation time of each parallel process under a given load distribution

Construct a cost function for load balancing using the following information

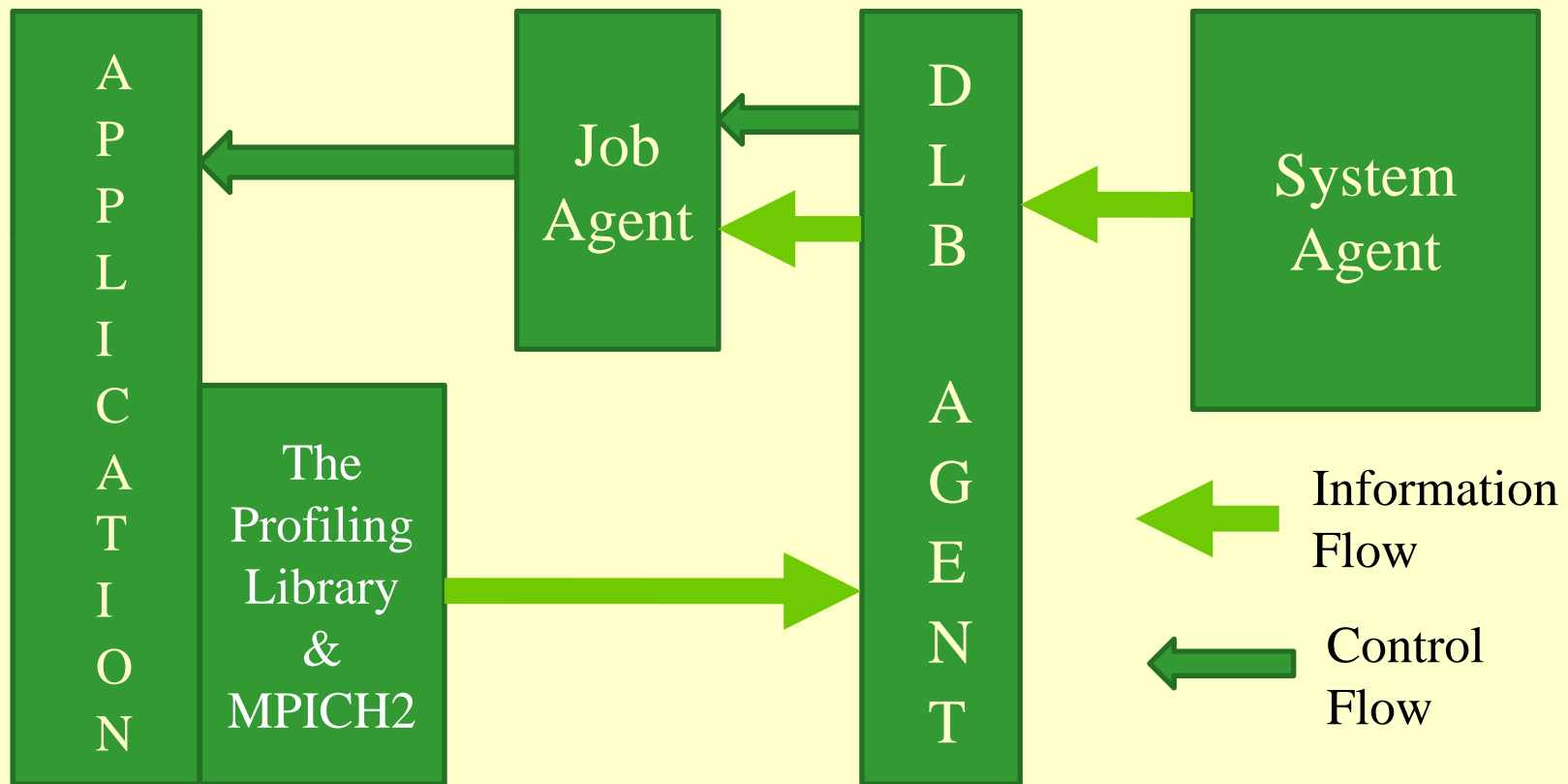
- The relative CPU core speed
- The workload of all processes of the parallel program
- The other computation load on each computer
- The communication speed between each pair of computers

Approach step 5

- Find a load distribution that minimizing the program execution time

Using greedy algorithm to find an optimal parallel load distribution based on the minimization of the cost function

Implementation of the approach – the DLB tool structure



DLB - System Agent

- A System Agent is executed on every computer all the time
- System Agent collects and maintains information of
 - Computer speed
 - Computer load
 - Computer owner's scheduler calendar
 - Network speed

DLB – DLB Agent

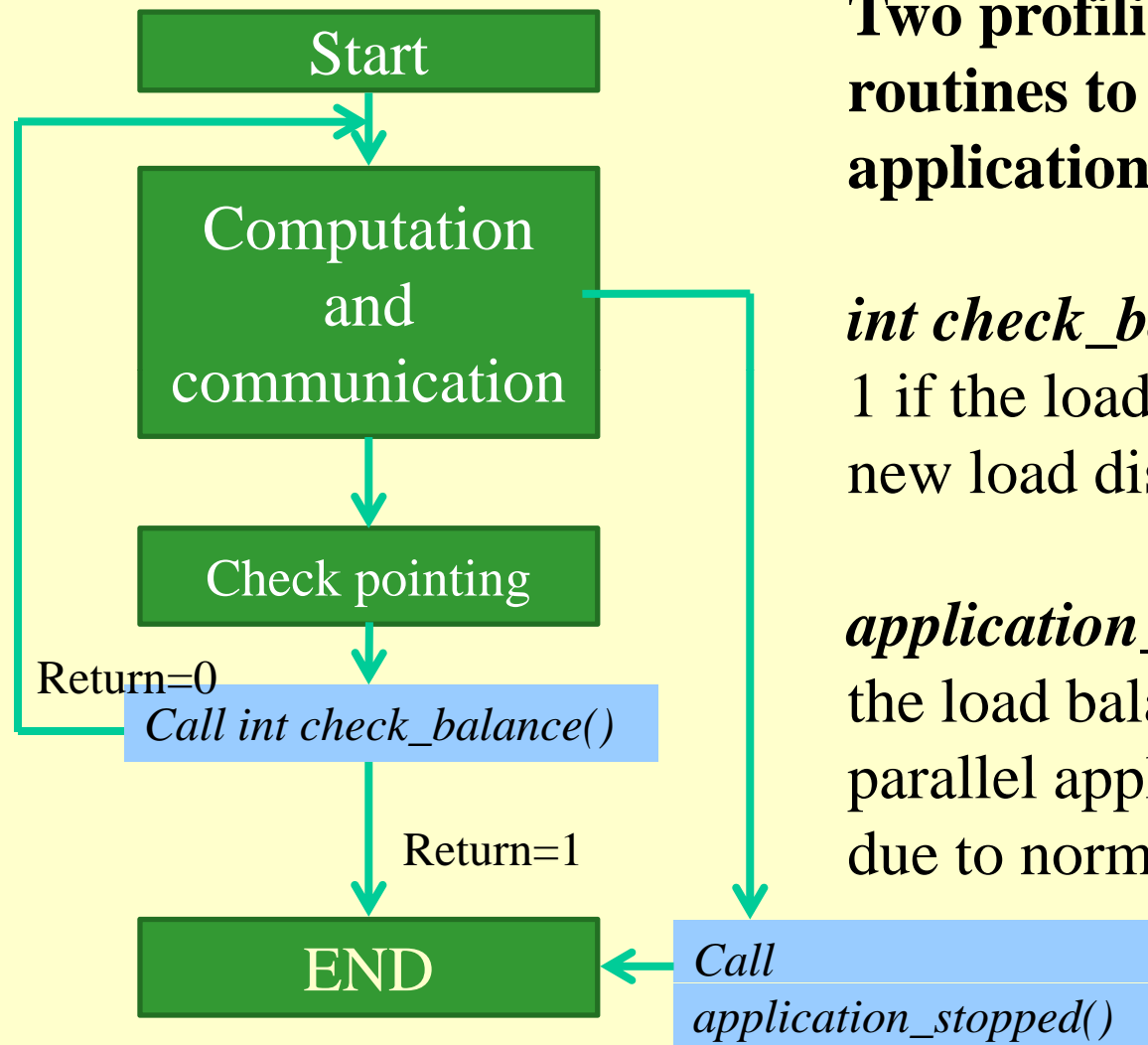
DLB agent is responsible for

- Gathering the information of all computers through System Agents
- Communicating with the application program through the Profiling Library in the application code
- Suggesting balanced load distribution to the Job Agent

DLB – Job Agent

- Job Agent is responsible for
 - Starting the parallel application according to DLB agent's suggestion
 - Monitor the execution status of each process of the parallel job
 - Report to DLB Agent if there is abnormality in parallel program execution

DLB - Application program preparation



Two profiling library routines to be inserted to the application program

int check_balance() - Returns 1 if the load balancer suggests new load distribution.

application_stopped() - Tells the load balancer that the parallel application is stopping due to normal completion.

How to determine a computer system is load balanced?

- The load balancing algorithm always suggests a load distribution.
- We need a measure to determine if the given load distribution is optimal.
- Claim: The load distribution is optimal if all the computers are busy all the time while executing a parallel job.

CPU usage percentage

$$\text{CPU Usage Percentage} = \frac{\sum_{l=1}^{\# \text{ of processes}} \text{CPU elapsed time of } l^{\text{th}} \text{ block}}{(\# \text{ of CPU cores}) * \text{Wall clock time}} * 100\%$$

- CPU usage percentage is a measure of how much CPU power is used in one computer
- It was assumed that all the CPU cores in a computer are identical
- CPU usage percentage can be calculated during the execution of the parallel program

System efficiency

To evaluate the system utilization of all hardware used for a parallel job, *System Efficiency* (SE) is defined as follows

$$SE = \frac{CP^1 \times CS^1 + CP^2 \times CS^2 + CP^3 \times CS^3 + \dots + CP^n \times CS^n}{CS^1 + CS^2 + CS^3 + \dots + CS^n}$$

where

CP^n = CPU utilization percentage of the nth computer,

CS^n = CPU relative speed of the nth PC.

Experiment - Parallel application program

- Parallel CFD test solver
 - A 3D transient heat equation implemented on structured grids with forward time central space differentiation method.
 - Can be downloaded from http://www.engr.iupui.edu/me/newmerl/cfdl_software.htm

Experiment - Job description

- One parallel Job
 - Parallel CFD test solver code
 - Number of equal sized blocks: 12
 - 2 blocks in X direction
 - 2 blocks in Y direction
 - 3 blocks in Z direction
 - Block size is approximately 8MB (15,625 nodes/block)
- To be run on 4 specific Windows based computers.

Experiment - Computer list and their relative speed

Table 1 PCs used in the experiment

PC Name	Benchmark Execution CPU Time (sec) use one CPU core	Number of CPU cores
in-engr-sl11133	8.234375	1
in-engr-sl11132	7.984375	1
in-engr-sl11134	8.465625	2
in-engr-sl11135	8.578125	2

Experiment - Initial load distribution

Table 2: Experimental results for the initial distribution

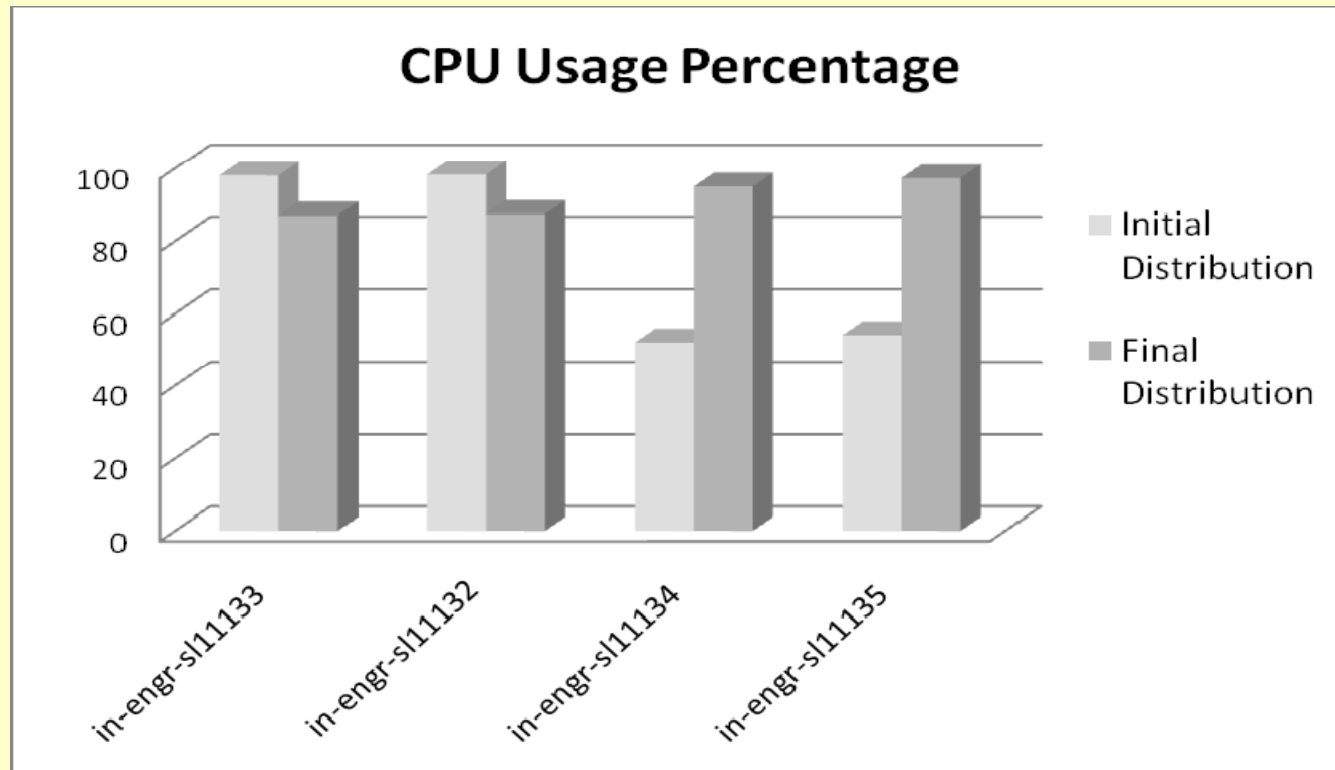
PC Name	Total Load	Blocks	Measured CPU Usage Percentage
in-engr-sl11133	3	1 2 3	98.024
in-engr-sl11132	3	4 5 6	98.265
in-engr-sl11134	3	7 8 9	52.127
in-engr-sl11135	3	10 11 12	54.168

Experiment - Load distribution after load balancing

Table 3. Experimental results of the final distribution

PC Name	Total Load	Blocks	CPU Usage percentage
in-engr-sl11133	2	1 2	87.014
in-engr-sl11132	2	3 4	87.352
in-engr-sl11134	4	5 6 7 8	94.914
in-engr-sl11135	4	9 10 11 12	97.299

Experiment- System efficiency



- Initial system efficiency = 68.64%
- Final system efficiency = 93.03%

Discussions

- This approach can only support the multi-core computers that all CPU cores share memory.
- This approach cannot support multi-core systems that use distributed memory and master/slave configuration (e.g., IBM cell processors).

Conclusion

- The dynamic load balancing tool is augmented to be support multi-core UNIX based and Microsoft Windows (XP and Vista) based computers.
- The tool is successfully tested for supporting a parallel CFD code.