What is Load Imbalance

- Irregular distribution of load among resources.
  - Resources can be: computational, network, processing units...

- Our target: MPI load Imbalance
  - MPI is the standard de facto in HPC applications
  - MPI processes do not share data
    - Moving data around is expensive
Load Imbalance: Magnitude of the tragedy

- Ideal
- 5% difference
- 5% difference in 1024 processes

Process 1 (55%)
Process 2 (45%)
Process 1025 (45%)
...

1s * 1024 CPUS = 1024 s = 17 minutes of CPU
17m * 10,000 time steps = 2,844 CPU hours

Load Imbalance: Measuring it

- Which application is more imbalanced?
  - A)
  - B)
  - C)
  - D)
Load Imbalance: Measuring it

Our focus is to make the most efficient use of computational resources.

Load Balance = \frac{\text{Useful CPU time}}{\text{Total used CPU time}} = \frac{\sum_{n=1}^{\text{numProcs}} (t_n)}{\text{Max}_n \cdot \text{numProcs} (t_n) \cdot \text{numProcs}} = \frac{\text{Average}_{n=1}^{\text{numProcs}} (t_n)}{\text{Max}_n \cdot \text{numProcs} (t_n)}

- numProcs = number of MPI processes
- t_n = execution time of process n
- 0 < LB < 1
- LB = 1 \rightarrow \text{Perfect Load Balance}

\[ \begin{align*}
\text{LB} &= \frac{3 + 1.5}{3 \cdot 2} = 0.75 \\
\text{UsedCPU} &= 3 \cdot 2 = 6
\end{align*} \]

Which application is more imbalanced?

\[ \begin{align*}
\text{A) } (7 \cdot 10) + 5 &= 75 \text{ ms} \\
\text{B) } (7 \cdot 5) + 10 &= 45 \text{ ms} \\
\text{C) } (4 \cdot 10) + (4 \cdot 5) &= 50 \text{ ms} \\
\text{D) } (10 + 5 + (6 \cdot 7.5)) &= 75 \text{ ms}
\end{align*} \]
Load Imbalance: Solution from developers?

- Expensive in terms of:
  - Computational resources
  - Personal resources
- What happens if we change the input?
- And the hardware?
- Is it a real solution?

Load Imbalance: Where?

- Bottleneck
- Infrastructure
- Preemption
- Software Variability
- Heterogeneous performance
- Hardware heterogeneity
- Computational load
- Data distribution

Running function

Cycles per µ second
Load Imbalance: Still searching for a solution...

- Different sources... different solutions
  - Data distribution
    - Redistribute → New Input, redistribute again?
  - Hardware heterogeneity
    - Tune specifically for architecture → New machine, tune again?
  - Infrastructure
    - Adapt code to infrastructure → New software or hardware, adapt again?
  - Software/Hardware variability
    - ???

- Our Solution: React when imbalance is happening
  - We can not fight it, lets adapt!
  - One solution to rule solve them all

Be water, my friend !!!

Bruce Lee

The DLB Library
Dynamic Load Balancing - DLB

- Our objectives:
  - Address all sources of imbalance
    - Fine Grain, dynamic...
      - How?
        - Detect imbalance at runtime
        - React immediately
  - Real product for HPC
    - Use common programming model/environment
      - MPI + OpenMP
  - Transparent to the application
    - Runtime library

The idea: Lend When Idle (LeWI)

- Load balance MPI processes within a computational node
  - Use computational resources of a process when not using them to speed up another process in the same node

- Original

- LeWI

Diagram: Lend When Idle (LeWI) process flow.
**LeWI: A image trace is worth a thousand words**

- **Original:**
  - 2x8

- **With LeWI:**
  - 2x8

---

**DLB: Main concepts**

- **CPU (core):** Minimum computing unit acknowledged by DLB, where one thread (and only one at the same time) can run.

- **Idle CPU:** A CPU that is not being used to do useful computation.

- **Owner:** Process that owns a CPU. A process owns the resources where it is started. A CPU can only be owned by one process at the same time.

- **Lend:** When the owner of a CPU is not using it, the CPU can be lent to the system. When a CPU is lent, a process that it is not its owner can use it.

- **Claim:** When the owner of a CPU wants to use it after lending it, the owner can claim the CPU.

- **Ask for Resources:** A process of the system can ask DLB for idle CPUs to speed up its execution.
DLB: How?

- Runtime library: DLB
  - Transversal to different layers of the software stack
  - Using standard mechanisms whenever possible
    - Facilitate the adoption without modifying existing codes
  - MPI:
    - Intercept MPI calls using PMPI standard interface
  - OpenMP:
    - Use standard OpenMP API
    - `omp_set_num_threads(x)`

P MPI: Profiling interface for MPI

- MPI libraries implement an internal interface (PMPI) that implements the MPI call code
- MPI calls can be redefined in a dynamic library
- The intercepting library is loaded when starting the application
  - `export LD_PRELOAD = libdlb_mpi.so`
  - The dynamically loaded library has preference
- Within the intercepted call the corresponding PMPI function must be called
PMPI Interception

- Using DLB and Extrae
  - Both use PMPI interface

- Integration:
  - Extrae intercepts MPI calls with PMPI
  - DLB API called from Extrae before and after each MPI call
  - DLB does not intercept MPI calls
    - `export LD_PRELOAD = libdlb_mpi_instr.so`

- And other profiling tools using PMPI?
  - We are studying using PnMPI
    - Allows n tools intercepting MPI
    - An order between them must be selected
    - All the tools must support PnMPI
    - So far no conflicts have been found... Future Work

MPI blocking mode

- MPI is greedy in the use of CPU
  - By default it will busy wait for messages/synchronizations to arrive
  - If the CPU is used by the MPI process waiting for the message we can not use it for doing useful computation by another thread.

- Different behavior for different MPI libraries 🤔

- We have two options:
  - Leave all the CPUs assigned to a process but one
    - `export DLB_ARGS += "--lewi-mpi=no"`
  - Tell MPI not to busy wait
    - `export I_MPI_WAIT_MODE=1`
    - `export DLB_ARGS += "--lewi-mpi"`
OpenMP: Malleability

- OpenMP is malleable, we can change number of threads
  - `omp_set_num_threads(int x)`
  - But only outside a parallel region

- But some programming practices can avoid malleability: 👎
  - Program in function of the thread Id
    - `omp_get_thread_num(int x)`
    - Fear if you see this call!

  - Do reductions “by hand”
    - Allocate memory in function of the number of threads and each one will reduce in its piece of data.

  - Avoid these practices please!
**OpenMP: Malleability**

- Use `omp_set_num_threads(x)`
  - It can only be called outside a parallel region (says the OpenMP standard)
  - Impact in DLB...

```
OMP PARALLEL
  OMP DO
  OMP DO
OMP PARALLEL DO
OMP PARALLEL DO
OMP PARALLEL DO
OMP PARALLEL DO
```

![Diagram of OpenMP parallel regions]

- Mth#1: The overhead of opening/closing parallelism

**OpenMP in DLB**

- Add a call to `int DLB_Borrow(void)` before each parallel
- `int DLB_Borrow(void)` will check the system for idle CPUs and update the number of threads in case necessary

```c
DLB_Borrow();
#pragma omp parallel do for (i=0; i<n; i++)
  
  compute...

...}
```

- This can be done by an automatic replacement in the code
- Latest news!
  - Working in using OMPT (tracing tool for OpenMP to appear in 5.0)
- Meanwhile...

```c
int DLB_Borrow(void){
  check_idle_cpus(x);
  set_omp_num_threads(x);
}
```
Integration with Nanos++

- Nanos++: Parallel Runtime developed at BSC
  - Implements OpenMP 4.5 and OmpSs
  - Forerunner for OpenMP

- Mercurium: Source to source compiler developed at BSC
  - Generates code for Nanos++

Integration with Nanos++

- There is no need to modify the application at all
  - The runtime will call the DLB API where necessary to ask for resources or return them

- Compile with Mercurium

- Run enabling DLB
  - Mandatory: `NX_ARGS+= "--enable-dlb --enable-block"
  - Recommended: `NX_ARGS+= "--force-tie-master"
  - In some cases: `NX_ARGS+= "--warmup-threads"

- Win!
More malleability with OmpSs

- OpenMP (Fork-join model)
  - OMP PARALLEL
- OmpSs(Task based)
  - Can adjust threads

Integration with Nanos++

- Taking advantage of the integration and increased OmpSs malleability
  - Threads are autonomous
    - Fast response
    - The master thread is not a bottleneck
    - Benefit from imbalances at OmpSs level too
Summing up to use DLB...

- export LD_PRELOAD = libdlb_mpi.so
- export DLB_ARGS = "--lewi"

If we want to use the CPU executing the MPI calls
  - export I_MPI_WAIT_MODE=1
  - export DLB_ARGS += "--lewi-mpi"

If we use Nanos++
  - NX_ARGS+= "--enable-dlb --enable-block"
  - NX_ARGS+= "--force-tie-master --warmup-threads"

else
  - Add DLB_Borrow() before each #pragma omp parallel

Multiple Applications

- We can share CPUs between different applications running in the same node
- Do not need MPI
- Transparent to the user, works out of the box
DROM
Dynamic Resource Ownership Management

API for superior entity
- Job Scheduler
- Resource manager
- User

Allow to change the assigned resources (CPUs) to a process

Some possible use cases:
- A) User wants to give more priority to one of the processes in the node
- B) Job scheduler wants to start a high priority app. using the resources allocated for another application
- C) Application is not using the resources in a node efficiently (i.e., the bottleneck is on another node) can free them to avoid accounting.
DROM: Use cases

A) User: Increase priority to App2

B) Job Scheduler: Run High priority App2 in resources assigned to App1

C) App1: Release 2 CPUs because not using efficiently

DROM: How to

A) $> dlb_taskset -p pid_app2 -c 0-5

B) $> dlb_taskset -c 0,1 ./App2

C) DLB_DROM_SetProcessMask(my_pid, [0,0,1,1]);
About DLB

- Current stable version 2.0 (January 2018)
  - LeWi
    - Full support of MPI.
    - Full support with NanoS runtime.
    - Support for OpenMP through API.
  - DROM
    - With OMPT support
  - Mode of communication with runtimes:
    - Asynchronous
    - Polling
    - Callback system: Ease of integration
  - New DLB API
    - Refactored
    - More exhaustive
    - More clear

- Free Download under LGPL-v3 license:
  https://pm.bsc.es/dlb-downloads

Work in Progress

- DROM
  - Implemented, evaluate performance

- OMPT
  - Enable use for any OpenMP runtime supporting OMPT (OpenMP 5.0)
  - Not “legal” according to the standard

- Study performance in many-core
  - i.e. Intel Xeon Phi KNL 256 threads

- Runtime Monitoring Tool
  - Monitor different levels and collect metrics
  - Offer an API to consult metrics during execution

- Load Balancing across containers
  - Studying feasibility, performance, issues and opportunities
  - Docker, Singularity...
Challenges

- Transversal to different layers, make the cooperate!!
  - MPI libraries are not willing to expose the non busy wait mode
    - They want all CPU cycles for them, but they are wasting them...
  - OS could help handling the cores? Giving priorities?

- Change mentality from “heroism programming” to trusting the runtime
  - Applications should stop doing things “by hand”
  - Let’s help them:
    - By addressing their needs and offering non intrusive solutions
    - By offering transversal solutions

- Malleability, malleability everywhere!!!
  - Application, Programming model, job scheduler...

FAQ
FAQ

- Why not “learn” and use previous redistribution?
- What about data locality?
- My application does not perform well with OpenMP
- What about load balance between nodes?
- Why not overload CPUS, it’s the same you do!
- How do you decide to which process CPUS go?
- I already have a load balancing algorithm within my application
- How do I know the different options in DLB?

Why not “learn” and use previous redistribution?

- There is a policy in DLB that does a “static” distribution of CPUs based in the load of each process
  - --policy=WEIGHT
    - Detects iterations, based in the MPI calls pattern
    - Computes an optimum distribution of CPUs
    - Applies it
    - Performance was much worse than LeWI → LeWI is more flexible
    - Code is deprecated

- Another policy that merge the functionality of WEIGHT and LeWI was implemented (Redistribute and Lend)
  - --policy=RaL
  - Performance was equal to the one obtained by LeWI

- We can recover these if we find the need
How do you decide to which process CPUS go?

- We do not decide it, it is first come, first served

- So far, our experience is: If there is a free CPU and some one willing to use it, do it.

- But... we might implement some accounting in the future if more actors come in... different apps, different users, different programming models...

- We DO decide which CPU to take first...

What about data locality?

- In some kernels spawning threads to another socket can have a penalty

- We can choose with flag `--lewi-affinity` in DLB_ARGS environment variable which CPU a process will acquire first when asking for resources,,
  - *any*: Take the first free CPU, does not take into account topology
  - *nearby-first*: Take first CPUs that are “affine” to me, and then the others
  - *spread-ifempty*: Take first CPUs that are affine to me, take CPUs from another socket only if all the CPUs in that socket are free (meaning no body is running there)
  - *nearby-only*: Take only CPUs that are affine to me
My application does not perform well/it is not parallelized with OpenMP

- Don’t worry!
- In fact usually it is the best configuration... gives more flexibility to DLB

What about load balance between nodes?

- We do not have any solution for this yet
- It is a quite different problem
  - Big difference in granularity, moving data between nodes is expensive
- But... good news is...
  - We are achieving very good results by balancing inside the node even when running up to 1024 nodes
I already have a load balancing algorithm within my application

- Does it solve this?
- Fine grain + system noise
- Blue lines original application (2 different runs)
- Red lines same run with DLB (2 different runs)

Clearly visible spikes without DLB are absorbed by DLB

How do I know the different options in DLB?

- `[DLB_HOME]/bin/dlb -help`

The library configuration can be set using arguments added to the DLB_ARGS environment variable. Possible options are listed below:

- `--lewi`: no (bool)
- `--drom`: no (bool)
- `--mode`: polling [polling, async]
- `--instrument`: yes (bool)
- `--instrument-counters`: no (bool)
- `--lewi-mpi`: no (bool)
- `--lewi-mpi-calls`: all [all, barrier, collectives]
- `--lewi-affinity`: nearby-first [any, nearby-first, nearby-only, spread-ifempty]
- `--lewi-greedy`: no (bool)
- `--lewi-warmup`: no (bool)
Thank you

marta.garcia@bsc.es
victor.lopez@bsc.es
https://pm.bsc.es/dlb