

Parallel Programming Using MPI

Prof. Hank Dietz

KAOS Seminar, February 8, 2012

University of Kentucky
Electrical & Computer Engineering

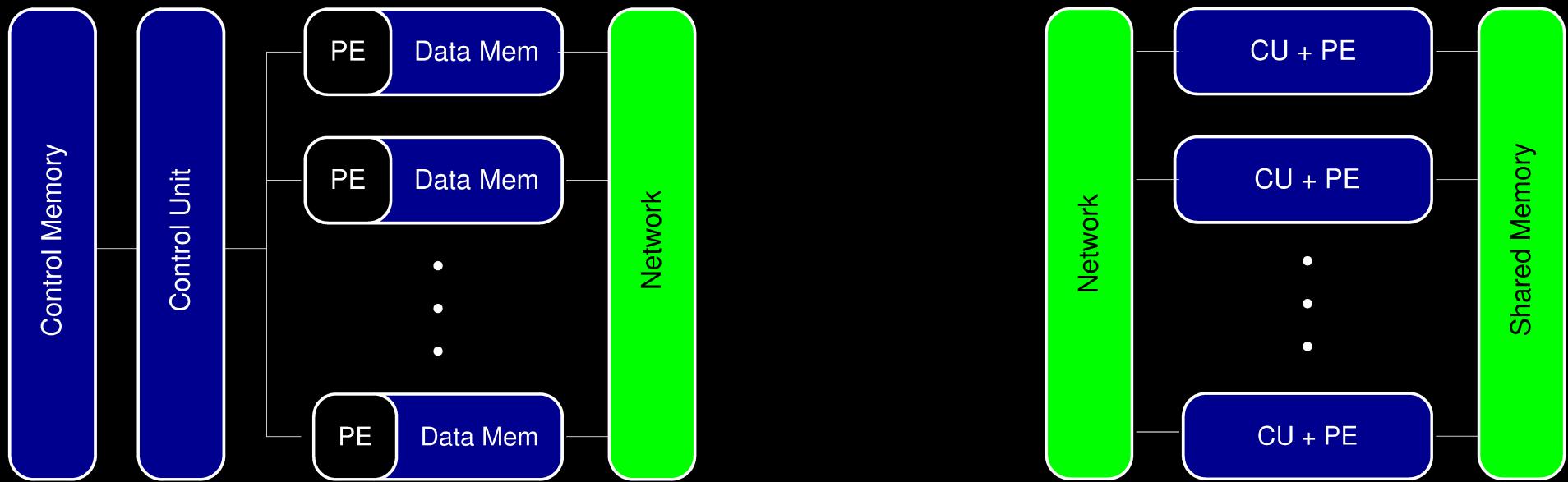
Parallel Processing

- Process N “pieces” simultaneously, get up to a factor of N speedup
- Parallel architectures:
 - Pipeline
 - Superscalar / VLIW / EPIC
 - SWAR (SIMD Within A Register)
 - SMP / Multi-core
 - GPU (Graphics Processing Unit)
 - Cluster / Farm / Grid / Cloud

Is Parallel Processing Easy?

- Some is **automatic** (by compiler/hardware), some is **SIMDish**, some is **MIMD**
- Parallel architectures:
 - **Pipeline**
 - **Superscalar / VLIW / EPIC**
 - **SWAR** (SIMD Within A Register)
 - **SMP / Multi-core**
 - **GPU** (Graphics Processing Unit)
 - **Cluster / Farm / Grid / Cloud**

SIMD Vs. MIMD



- SIMD: 1 PC, efficient, synchronous
- MIMD: N PCs, flexible, autonomous PEs

Memory In MIMDs

- Shared everything – threads
- Shared something – shared/private data
- Shared nothing – distributed memory
- Sharing can be via hardware or software;
e.g., DSM uses page faults to move shared
data where it is needed

Shared Nothing Models

- Can be efficient on all flavors of MIMD
- Debugging is relatively easy
- Generally communicate via messages, which are explicitly sent/received by PEs
- Lots of programming environments:
 - Languages (nothing very popular)
 - TCP/UDP Sockets
 - Message-passing libraries

Message-Passing Libraries

- Lots – every vendor had at least one...
- PVM (Parallel Virtual Machine)
- MPI (Message Passing Interface)
 - A portable, official, standard
 - More efficient than PVM
 - Very (**too?**) rich set of operations
 - Various add-ons, including basic user interface for running MPI programs

MPI Basics

- **Fortran, C, etc. bindings**
- Three separate ways to communicate:
 - Messages (blocking or non-blocking)
 - Collective communications
 - RMA (Remote Memory Access), aka, one-sided messages
- Run-time user interface stuff
- Many implementations, we use OpenMPI

Compute Pi Using Serial C

```
#include <stdlib.h>
#include <stdio.h>

main(int argc, char **argv)
{
    double width, sum; int intervals, i;

    intervals = atoi(argv[1]);
    width = 1.0 / intervals;
    sum = 0;
    for (i=0; i<intervals; ++i) {
        double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
    }
    sum *= width;

    printf("Pi is approximately %14.12f\n", sum);
    return(0);
}
```

MPI Collectives

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
    double width, sum, lsum; int intervals, i, nproc, iproc;

    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervals = atoi(argv[1]);
    width = 1.0 / intervals;
    lsum = 0;
    for (i=iproc; i<intervals; i+=nproc) {
        double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
    }
    lsum *= width;

    MPI_Reduce(&lsum, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (iproc == 0) {
        printf("Pi is approximately %14.12f\n", sum);
    }

    MPI_Finalize();
    return(0);
}
```

MPI Messages (blocking)

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
    double width, sum, lsum; int intervals, i, nproc, iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervals = atoi(argv[1]);
    width = 1.0 / intervals;
    lsum = 0;
    for (i=iproc; i<intervals; i+=nproc) {
        double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
    }
    lsum *= width;

    if (iproc != 0) {
        MPI_Send(&lsum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    } else {
        sum = lsum;
        For (i=1; i<nproc; ++i) {
            MPI_Status status;
            MPI_Recv(&lsum, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
                     MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            sum += lsum;
        }
        printf("Pi is approximately %14.12f\n", sum);
    }

    MPI_Finalize();
    return(0);
}
```

MPI Remote Memory Access

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
    double width, sum, lsum; int intervals, i, nproc, iproc;
    MPI_Win sum_win;

    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    MPI_Win_create(&sum, sizeof(sum), sizeof(sum),
                  0, MPI_COMM_WORLD, &sum_win);
    MPI_Win_fence(0, sum_win);
    intervals = atoi(argv[1]);
    width = 1.0 / intervals;
    lsum = 0;
    for (i=iproc; i<intervals; i+=nproc) {
        double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
    }
    lsum *= width;

    MPI_Accumulate(&lsum, 1, MPI_DOUBLE, 0, 0,
                  1, MPI_DOUBLE, MPI_SUM, sum_win);
    MPI_Win_fence(0, sum_win);
    if (iproc == 0) {
        printf("Pi is approximately %14.12f\n", sum);
    }

    MPI_Finalize();
    return(0);
}
```

OpenMPI User Interface

- **mpicc** – wraps C compiler, with options
 - Looks just like underlying C compiler...
- **mpirun** – run an MPI program, e.g.:
mpirun -n PEs -hostfile nodelist
- For OpenMPI, various **ompi** and **orte** ...

More Info

- MPI versions:
1.0, 1.1, 1.2, 1.3, 2.0, 2.1, 2.2
- Lots of MPI stuff out there...
 - <http://www.mpi-forum.org/>
 - <http://www.open-mpi.org/>
 - <http://computing.llnl.gov/tutorials/mpi/>