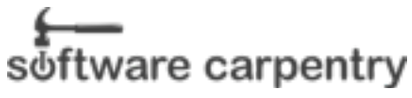
Conventional parallelism models

Presented by Alex Kosenkov



Agenda

- MPI
- pthreads
- OpenMP

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

software carpentry

MPI

- Message Passing (Application Programming) Interface
- Distributed memory systems, NUMA Friendly

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

software carpentry

MPI: Initialisation

```c
#include <mpi.h>
#include <stdio.h>
#include <stdexcept>

int main (int argc, char** argv){
    int rank, size;

    MPI_Init(&argc, &argv); // MPI_THREAD_SINGLE
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf( "Hello world from process %d of %d\n", rank, size );

    MPI_Finalize();
    return 0;
}

int main (int argc, char** argv){
    int rank, size;
    int level, req = MPI_THREAD_MULTIPLE; // or MPI_THREAD_SERIALIZED, MPI_THREAD_FUNNELED

    MPI_Init_thread(&argc, &argv, req, &level);
    if(level != req) throw std::runtime_error("Error: couldn't provide desired threading level\n");
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf( "Hello world from process %d of %d\n", rank, size );

    MPI_Finalize();
    return 0;
}
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

MPI: Point-to-point

- Synchronous
- Asynchronous
- Ready
- Buffered
- Persistent

MPI: Point-to-point (synchronous)

```
int MPI_Send(const void* buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);
int MPI_Rsend(...);

int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status* status);
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

MPI: Point-to-point (asynchronous)

```
int MPI_Isend(const void* buf, int count, MPI_Datatype datatype,
              int dest, int tag, MPI_Comm comm, MPI_Request* req);
int MPI_Irsend(...);

int MPI_Irecv(void* buf, int count, MPI_Datatype datatype,
              int source, int tag, MPI_Comm comm, MPI_Request* req);
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

MPI: Point-to-point

Example: p2p, single message

```
 1 #include <mpi.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <stdexcept>
 5 #include <string>
 6 #include <memory.h>
 7
 8 void usecase_p2p(int rank){
 9     char* msg = (char*)calloc(10,sizeof(char));
10     if(rank == 0){
11         std::string content("Who is it");
12         memcpy(msg, content.c_str(), content.length());
13         MPI_Send(msg, 10, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
14     }else if(rank == 1){
15         MPI_Recv(msg, 10, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
16         printf("Received message: %s\n", msg);
17     }
18 }
19
20 int main (int argc, char** argv){
21     int rank, size;
22     MPI_Init(&argc, &argv);
23     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
24     MPI_Comm_size(MPI_COMM_WORLD, &size);
25     usecase_p2p(rank);
26     MPI_Finalize();
27     return 0;
28 }
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

MPI: Synchronisation / Progress

```
int MPI_Barrier(MPI_Comm comm);

int MPI_Wait(MPI_Request* request, MPI_Status* status);

int MPI_Waitall(int count, MPI_Request requests[], MPI_Status* status);

int MPI_Test(MPI_Request* request, int* flag, MPI_Status* status);

int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status* status);

int MPI_Iprobe(int source, int tag, MPI_Comm comm, int* flag, MPI_Status* status);
```

MPI: Synchronisation / Progress

Example: async p2p, wait and barrier

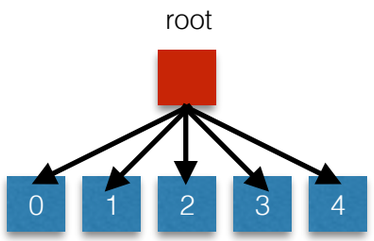
```
 1 #include <mpi.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string>
 5 #include <memory.h>
 6
 7 void usecase_p2p_async(int rank){
 8     char* msg = (char*)calloc(10,sizeof(char));
 9     MPI_Request requests[10];
10     if(rank == 0){
11         std::string content("Who is it");
12         memcpy(msg, content.c_str(), content.length());
13         for(int i = 0; i < 10; i++) MPI_Isend(&msg[i], 1, MPI_CHAR, 1, 0, MPI_COMM_WORLD, &requests[i]);
14         MPI_Waitall(10, requests, MPI_STATUSES_IGNORE);
15     }else if(rank == 1){
16         for(int i = 0; i < 10; i++) MPI_Irecv(&msg[i], 1, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &requests[i]);
17         MPI_Waitall(10, requests, MPI_STATUSES_IGNORE);
18     }
19     MPI_Barrier(MPI_COMM_WORLD);
20     if(rank == 1) printf("Received partial message: %s\n", msg);
21 }
22
23 int main (int argc, char** argv){
24     int rank, size;
25     MPI_Init(&argc, &argv);
26     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
27     MPI_Comm_size(MPI_COMM_WORLD, &size);
28     usecase_p2p_async(rank);
29     MPI_Finalize();
30     return 0;
31 }
```

MPI: Collectives

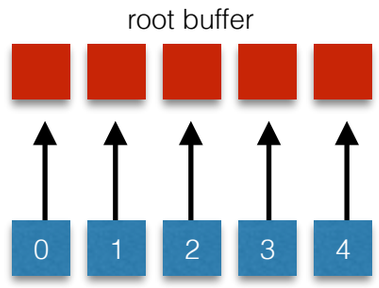
```c
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm);

int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf,
               int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf,
                int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

int MPI_Alltoall(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf,
                 int recvcount, MPI_Datatype recvtype, MPI_Comm comm);

int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
               MPI_Op op, int root, MPI_Comm comm);
```

MPI: Collectives

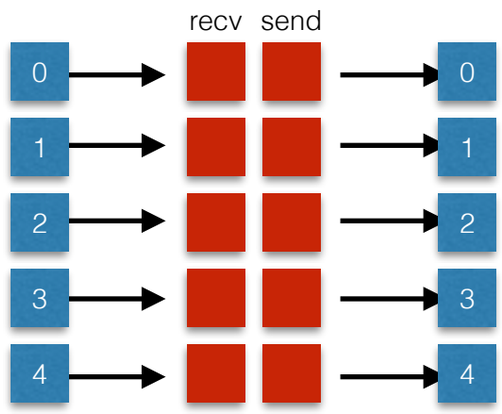
Broadcast



Gather

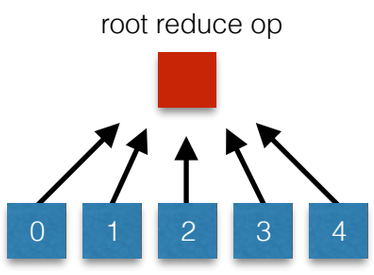


All to all

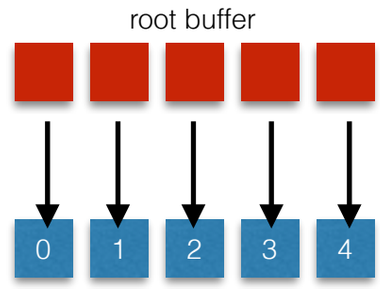


* at every rank

Reduce



Scatter



MPI: Collectives

MPI reductions ops (MPI_Op):

```
MPI_MAX             maximum
MPI_MIN             minimum
MPI_SUM             sum
MPI_PROD            product
MPI_LAND            logical and
MPI_BAND            bit-wise and
MPI_LOR             logical or
MPI_BOR             bit-wise or
MPI_LXOR            logical xor
MPI_BXOR            bit-wise xor
MPI_MAXLOC          max value and location
MPI_MINLOC          min value and location
```

MPI: Collectives

Example: gather message

```
 1 #include <mpi.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string>
 5 #include <memory.h>
 6
 7 char generate(int rank){
 8     switch(rank){
 9         case 0 : return 'W';
10         case 1 : return 'h';
11         case 2 : return 'o';
12         case 3 : return ' ';
13         case 4 : return 'i';
14         case 5 : return 's';
15         case 6 : return ' ';
16         case 7 : return 'i';
17         case 8 : return 't';
18     }
19     return 'Z';
20 }
21
22 void usecase_gather(int rank, int size){
23     char* msg = (char*)calloc(size+1,sizeof(char));
24     char chunk = generate(rank);
25     MPI_Gather(&chunk, 1, MPI_CHAR, msg, 1, MPI_CHAR, 0, MPI_COMM_WORLD);
26     if(rank == 0) printf("Gathered message: %s\n", msg);
27 }
28
29 int main (int argc, char** argv){
30     int rank, size;
31     MPI_Init(&argc, &argv);
32     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
33     MPI_Comm_size(MPI_COMM_WORLD, &size);
34     usecase_gather(rank,size);
35     MPI_Finalize();
36     return 0;
37 }
```

MPI: Collectives

Example: alltoall message (allgather)

```
 1 #include <mpi.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string>
 5 #include <memory.h>
 6
 7 char generate(int rank){
 8     switch(rank){
 9         case 0 : return 'W';
10         case 1 : return 'h';
11         case 2 : return 'o';
12         case 3 : return ' ';
13         case 4 : return 'i';
14         case 5 : return 's';
15         case 6 : return ' ';
16         case 7 : return 'i';
17         case 8 : return 't';
18     }
19     return 'Z';
20 }
21
22 void usecase_alltoall(int rank, int size){
23     char* msg_global = (char*)calloc(size+1,sizeof(char));
24     char* msg_local = (char*)calloc(size+1,sizeof(char));
25     char chunk = generate(rank);
26     memset(msg_local, chunk, (size+1)*sizeof(char));
27     MPI_Alltoall(msg_local, 1, MPI_CHAR, msg_global, 1, MPI_CHAR, MPI_COMM_WORLD);
28     for(int i = 0; i < size; i++){
29         MPI_Barrier(MPI_COMM_WORLD);
30         if(rank == i) printf("Gathered message: %s\n", msg_global);
31     }
32 }
33
34 int main (int argc, char** argv){
35     int rank, size;
36     MPI_Init(&argc, &argv);
37     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
38     MPI_Comm_size(MPI_COMM_WORLD, &size);
39     usecase_alltoall(rank,size);
40     MPI_Finalize();
41     return 0;
42 }
```

MPI: Collectives

```
int MPI_Gatherv(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf,
                const int recvcounts[], const int displs[], MPI_Datatype recvtype,
                int root, MPI_Comm comm);

int MPI_Scatterv(const void *sendbuf, const int sendcounts[], const int displs[],
                 MPI_Datatype sendtype, void *recvbuf, int recvcount,
                 MPI_Datatype recvtype, int root, MPI_Comm comm);

int MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf,
                  int recvcount, MPI_Datatype recvtype, MPI_Comm comm);

int MPI_Allreduce(const void *sendbuf, void *rcvbuf, int count, MPI_Datatype datatype,
                  MPI_Op op, MPI_Comm comm);
```

Note: each collective has an asynchronous version in MPI-3 (..., MPI_Request request)*

MPI: Derived Data types

```
int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype);

int MPI_Type_vector(int count, int blocklength, int stride,
                    MPI_Datatype old, MPI_Datatype *new);

int MPI_Type_indexed(int count, const int blocklengths[], const int displacements[],
                     MPI_Datatype old, MPI_Datatype *new);

int MPI_Type_struct(int count, int *array_of_blocklengths, MPI_Aint *displacements,
                    MPI_Datatype *types, MPI_Datatype *new);

int MPI_Type_commit(MPI_Datatype *datatype);

int MPI_Type_free(MPI_Datatype *datatype);
```

MPI: Derived Data types

Example: constructing datatype

```
39      inline void packet_t::construct(int code, int count, const int* sizes, const MPI_Datatype* types){
40          MPI_Aint extent;
41          this->t_code = code;
42          this->compounds = (MPI_Datatype*)realloc(this->compounds, (this->count+count)*sizeof(MPI_Datatype));
43          this->displacements = (MPI_Aint*)realloc(this->displacements, (this->count+count)*sizeof(MPI_Aint));
44          this->sizes = (int*)realloc(this->sizes, (this->count+count)*sizeof(int));
45          memcpy(&(this->sizes[this->count]), sizes, count*sizeof(int));
46          memcpy(&(this->compounds[this->count]), types, count*sizeof(MPI_Datatype));
47
48          for(int i = 0; i < count; i++)
49          {
50              this->displacements[this->count + i] = this->t_size;
51              MPI_Type_extent(types[i], &extent);
52              this->t_size += extent*sizes[i];
53          }
54          this->count += count;
55      }

...

96      inline void packet_t::commit(){
97          MPI_Type_create_struct(this->count,           // count - number of types
98                                 this->sizes,           // number of instances of the same type (blocklengths)
99                                 this->displacements,   // starting positions
100                                this->compounds,       // types being used
101                                &this->mpi_t);
102
103         MPI_Type_commit(&this->mpi_t);
104     }
```

pthreads: overview

```
int pthread_create(pthread_t* thread, const pthread_attr_t* attr,
                   void* (*start_routine)(void*), void* arg);

int pthread_join(pthread_t thread, void** retval);

int pthread_mutex_init(pthread_mutex_t* mutex, const pthread_mutexattr_t* attr);

int pthread_mutex_destroy(pthread_mutex_t* mutex);

int pthread_mutex_lock(pthread_mutex_t* mutex);

int pthread_mutex_trylock(pthread_mutex_t* mutex);

int pthread_mutex_unlock(pthread_mutex_t* mutex);
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

OpenMP: overview

Pragmas:

```
#pragma omp parallel

#pragma omp for

#pragma omp task

#pragma omp taskwait

#pragma omp critical

#pragma omp barrier

#pragma omp single
```

Enviornment variables:

```
OMP_NUM_THREADS

OMP_NESTED

OMP_SCHEDULE

OMP_STACKSIZE

OMP_WAIT_POLICY

OMP_PLACES (4.0)
```

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3
 4 #define N 100000
 5
 6 int main(){
 7     double a[N];
 8
 9     #pragma omp parallel
10     printf("Thread number %d\n", omp_get_thread_num());
11
12     #pragma omp parallel
13     {
14         #pragma omp for
15         for(int i = 0; i < N; i++){
16             a[i] = i*i;
17         }
18     }
19     return 0;
20 }
```

```
~/Dropbox/sandbox $ g++ omp0.cpp -fopenmp
~/Dropbox/sandbox $ icpc omp0.cpp -openmp

~/Dropbox/sandbox $ OMP_NUM_THREADS=4 ./a.out
Thread number 3
Thread number 2
Thread number 1
Thread number 0
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

OpenMP: data ownership

- default
- shared (default)
- private
- firstprivate
- lastprivate
- threadprivate
- reduction
- copyin
- copyprivate
- linear

```
1  #include <omp.h>
2  #include <stdio.h>
3  #include <iostream>
4  #define N 1000
5
6  int main(){
7
8      int tid;
9      int threads_count = 0;
10     double sum = 0;
11
12     #pragma omp parallel private(tid) shared(threads_count)
13     {
14         tid = omp_get_thread_num();
15         #pragma omp critical
16         {
17             std::cout << "Thread number " << tid << "\n";
18             threads_count++;
19         }
20         #pragma omp single nowait
21         {
22             std::cout << "Single " << omp_get_thread_num() << "\n";
23         }
24     }
25     #pragma omp parallel for reduction(+:sum) schedule(static, 1)
26     for(int i = 0; i < N; i++) sum++;
27
28     return 0;
29 }
```

OpenMP: scheduling

schedule(*type*, *chunk*):

- static
- dynamic
- guided
- auto
- runtime

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <iostream>
 4 #include <vector>
 5 #include <algorithm>
 6
 7 #define N 10000
 8 #define CHUNK 10
 9
10 int main(){
11     std::vector<double> a(N*N);
12     std::vector<double> b(N);
13     std::vector<double> c(N);
14
15     auto generate = [](double& e){ e = std::rand(); };
16     std::for_each(a.begin(), a.end(), generate);
17     std::for_each(b.begin(), b.end(), generate);
18
19     double t1 = omp_get_wtime();
20
21     #pragma omp parallel for schedule(dynamic, CHUNK)
22     for(int i = 0; i < N; i++)
23     for(int j = 0; j < N; j++)
24         c[i] += a[i+j*N]*b[j];
25
26     std::cout << "Time elapsed: " << (omp_get_wtime()-t1) << "\n";
27     return 0;
28 }
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

OpenMP: run-time environment

```c
// OMP_NUM_THREADS
int omp_get_num_threads(void);
void omp_set_num_threads(int num_threads);

// OMP_NESTED
int omp_get_nested(void);
void omp_set_nested(int nested);

// OMP_SCHEDULE
void omp_get_schedule(omp_sched_t *kind, int *modifier);
void omp_set_schedule(omp_sched_t kind, int modifier);

// OMP_PROC_BIND
omp_proc_bind_t omp_get_proc_bind(void); // (4.0)
```

```c
// OMP_DYNAMIC
int omp_get_dynamic(void);
void omp_set_dynamic(int dynamic_threads);

// OMP_MAX_ACTIVE_LEVELS
int omp_get_max_active_levels(void);
void omp_set_max_active_levels(int max_levels);

// OMP_THREAD_LIMIT
int omp_get_thread_limit(void);

// OMP_CANCELLATION (4.0)
int omp_get_cancellation(void);

// OMP_DEFAULT_DEVICE (4.0)
int omp_get_default_device(void);
void omp_set_default_device(int device_num);
```

OpenMP: 4.0 additions

```
#pragma omp teams

#pragma omp simd

#pragma omp distribute

#pragma omp target

#pragma omp target update

#pragma omp cancel

#pragma omp cancellation point

#pragma omp declare reduction
```

Example: target device / map

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for(int i = 0; i < N; i++)
    B[i] *= 2;
```

Example: target device / map

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for(int i = 0; i < N; i += bsize)
    #pragma omp parallel for
    for (int b = i; b < i+bsize; b++)
        B[b] += 2;
```

Example: simd usage

```
float sum = 0.0f;
float *p = a;
int step = 4;
#pragma omp simd reduction(+:sum) linear(p:step)
for(int i = 0; i < N; ++i){
    sum += *p;
    p += step;
}
```

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

software carpentry

OpenMP: misc

```
#pragma omp sections
#pragma omp master
#pragma omp taskyield
#pragma omp taskgroup
#pragma omp atomic
#pragma omp flush
#pragma omp ordered

int omp_get_num_devices(void);
int omp_get_num_teams(void);
int omp_get_team_num(void);
int omp_is_initial_device(void);
```

```
int omp_get_max_threads(void);
int omp_get_num_procs(void);
int omp_in_parallel(void);
int omp_get_level(void);
int omp_get_ancestor_thread_num(int level);
int omp_get_team_size(int level);
int omp_get_active_level(void);
int omp_in_final(void);

void omp_init_lock(omp_lock_t *lock);
void omp_destroy_lock(omp_lock_t *lock);
void omp_set_lock(omp_lock_t *lock);
void omp_unset_lock(omp_lock_t *lock);
int omp_test_lock(omp_lock_t *lock);
```

Questions

End of the third part

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

software carpentry