EPiGRAM

Exascale ProGRAmming Models

A new thread support level for hybrid programming with MPI endpoints

EASC 2015

Dan Holmes, Mark Bull, Jim Dinan

dholmes@epcc.ed.ac.uk, markb@epcc.ed.ac.uk, james.dinan@intel.com

Exascale – hardware trends

- Hardware design driven by power limits
- Hardware increasingly has deep hierarchy
  - Nodes -> CPUs -> cores -> SIMD vectors
  - Nodes -> GPUs -> SMs -> cores -> warps
- Hardware increasingly has varied locality
  - Multiple levels of NUMA regions
  - Multiple data paths: cache-coherency, RDMA
  - Moving data from “far away” to “near” is costly

Exascale – hybrid programming

- Unclear if pure MPI can scale to exascale
- Exploration of MPI+X, i.e. hybrid program
- Many options for X (MPI, OpenMP, PGAS)
- Most options for X involve OS threads
- Interoperability between MPI and threads?
  - Good: thread support defined in MPI Standard
  - Bad: multi-thread support has high overheads
  - Ugly: addressability, i.e. identifying threads

EPiGRAM

An MPI process is a logical PE

- MPI defines a flat hierarchy of processing elements: MPI processes
- This is a programming model concept
  - An MPI process is defined only by MPI function calls and their semantics
- It is not a programming system construct
  - c.f. OS process or POSIX thread
- Increasingly MPI processes are a bad model for complex hardware

EPiGRAM

Current thread support in MPI

- Specified in External Interfaces chapter
  - Threads & OS processes are external to MPI
  - Focused on implications for MPI library writers
- Four thread support levels:
  - `MPI_THREAD_SINGLE`
    - Only one thread will exist
  - `MPI_THREAD_FUNNELLED`
    - Only one thread will access MPI
  - `MPI_THREAD_SERIALIZED`
    - Only one thread will access MPI at one time
  - `MPI_THREAD_MULTIPLE`
    - No restrictions: any thread(s) can access MPI any time

EPiGRAM

Why have several levels?

- Implementation methods for MPI process concept impose practical considerations
  - Multiple: MPI must be fully thread-safe, protection of shared state “requires locks”
  - Serialised: MPI does not need to protect shared state from concurrent accesses
  - Funnelled: MPI can use thread-local features
  - Single: MPI free to use code even if it is not thread-safe

EPiGRAM

Threading issues for users 1

- Collectives: manual coding of hierarchy
  - E.g. each OpenMP thread provides a reduction value
  - All threads want to call `MPI_ALLREDUCE`
  - But only one thread allowed in MPI collective
  - So first, do OpenMP parallel reduction
    - Adds thread synchronisation overhead
  - Then, MPI reduction in OpenMP single region
    - Adds thread load-imbalance overhead

EPiGRAM

Threading issues for users 2

- Point-to-point: addressing each thread requires different tags or communicators
  - Communicators: requires too many for full addressability or general connectivity
  - Tags: no way for MPI to know relationship between tags and threads => shared-state
    - Single shared unexpected send queue
    - Single shared unmatched receive queue
    - Access must be serialised, by user or by MPI

EPiGRAM

Summary of MPI endpoints

- Additional logical PEs – MPI “processes”
  - Hierarchically associated with an MPI process
  - Addressable by rank in the group of a new communicator
  - Act and react like normal MPI processes
    - Except for `MPI_INIT_THREAD` & `MPI_FINALIZE`
  - Array of communicator handles returned by communicator creation function
    - Each returns a different `MPI_COMM_RANK` value

EPiGRAM

Modelling advantages

- Processing elements that share an address-space can be modelled in the application code using MPI endpoints
- Flexible; threads communicate via MPI

```
#pragma omp parallel
MPI_Comm_create_endpoints(numThreads);
#pragma omp for
for () {do_calc();
MPI_Neighbourhood_alltoall();}
```


EPiGRAM

New Optimisations Possible

- Shared state can be divided per endpoint
  - Example provided by proxy job demonstrator
- Dedicated resources for each endpoint
  - Separate queues per endpoint
  - {communicator, target rank} identifies context
  - If only one thread ever uses an endpoint then
    - the endpoint usage like “funnelled” definition
  - If only one thread at a time uses an endpoint
    - the endpoint usage like “serialised” definition

EPiGRAM

Application behaviour restriction

- One MPI endpoint per OS thread anticipated to be a common use case
- How does user inform MPI the application will restrict usage of threads & endpoints?
  - New INFO key supplied to communicator creation function
  - Specifying new thread support levels
  - Enhancing existing thread support levels

EPiGRAM

Hinting at behaviour restriction

- Current INFO keys are hints to the MPI
  - MPI library can ignore all INFO keys
  - MPI library not allowed to modify semantics based on INFO key information
  - User can lie!
  - MPI must check hint accuracy
- MPI would still need to be initialised with `MPI_THREAD_MULTIPLE`
- Considered but discounted

EPiGRAM

New thread support levels 1

- Additional thread support levels to extend (ideas that pre-date endpoints proposal)
  - `MPI_THREAD_AS_RANK`
    - Each thread calls `MPI_INIT` and becomes an MPI process with its own rank in `MPI_COMM_WORLD` (like "single" but excludes thread unsafe code)
  - `MPI_THREAD_FILTERED`
    - Some threads call `MPI_INIT` and become MPI processes, others delegate calling of MPI functions to an initialised thread (similar to "funnelled")

EPiGRAM

New thread support levels 2

- Additional thread support levels to extend definition of `MPI_THREAD_FUNNELED`
  - `MPI_THREAD_FILTERED`
    - One thread calls `MPI_INIT` and is MPI processes
    - MPI process creates multiple endpoints
    - All threads can call MPI at the same time if they all use different endpoints
    - Only one thread can use any particular endpoint

EPiGRAM

New thread support levels 3

- Additional thread support levels to extend definition of `MPI_THREAD_SERIALIZED`
  - `MPI_THREAD_SERIAL_EP`
    - One thread calls `MPI_INIT` and is MPI process
    - MPI process creates multiple endpoints
    - All threads can call MPI at the same time if they all use different endpoints
    - Any thread can use any endpoint but only one thread can use any particular endpoint at a time

EPiGRAM

Alter old thread support levels

- Add "per endpoint" wording to funnelled and serialised definitions
  - Backward compatible because all existing MPI processes have exactly one MPI endpoint with no possibility to create more
  - Intention is clear; precise wording is still undergoing active discussion
  - Funnelled definition linked to definition of “main thread”; should now be “main threads”?

EPiGRAM

Suggested MPI Standard Text 1

- `MPI_THREAD_FUNNELED`
  - The process may be multi-threaded, but the application must ensure that only the main thread makes MPI calls (for the definition of main thread, see `MPI_IS_THREAD_MAIN`).
- Main thread
  - The thread that called `MPI_INIT` or `MPI_INIT_THREAD` **or first uses a communicator handle returned by `MPI_COMM_CREATE_ENDPOINTS`**

EPiGRAM

Suggested MPI Standard Text 2

- `MPI_THREAD_SERIALIZED`
  - The process may be multi-threaded, and multiple threads may make **concurrent** MPI calls, but only one at a time **per endpoint**: MPI calls **using a single endpoint** are not made concurrently from two distinct threads (all MPI calls **for each endpoint** are "serialized").

EPiGRAM

Implementation Issues

- `MPI_COMM_CREATE_ENDPOINTS` can generate `MPI_ERR_ENDPOINTS`
  - if it cannot create/support the required number of endpoints (in existing proposal)
- For new “funnelled” and “serialised” levels
  - this will happen when MPI cannot provide isolated, dedicated resources for each new endpoint and cannot silently degrade to “multiple”-like implementation

EPiGRAM

Summary

- MPI endpoints introduces hierarchy of logical PEs that share an address-space
  - Enables flexible new mappings of logical PEs (MPI processes/endpoints) to physical PEs (OS processes/threads)
- Modifying “funnelled” & “serialised” levels
  - Extends their usefulness to more threads
  - Delays the time when “multiple” is needed
  - Backwards compatible

EPiGRAM