SINGLE-SIDED PGAS COMMUNICATIONS LIBRARIES

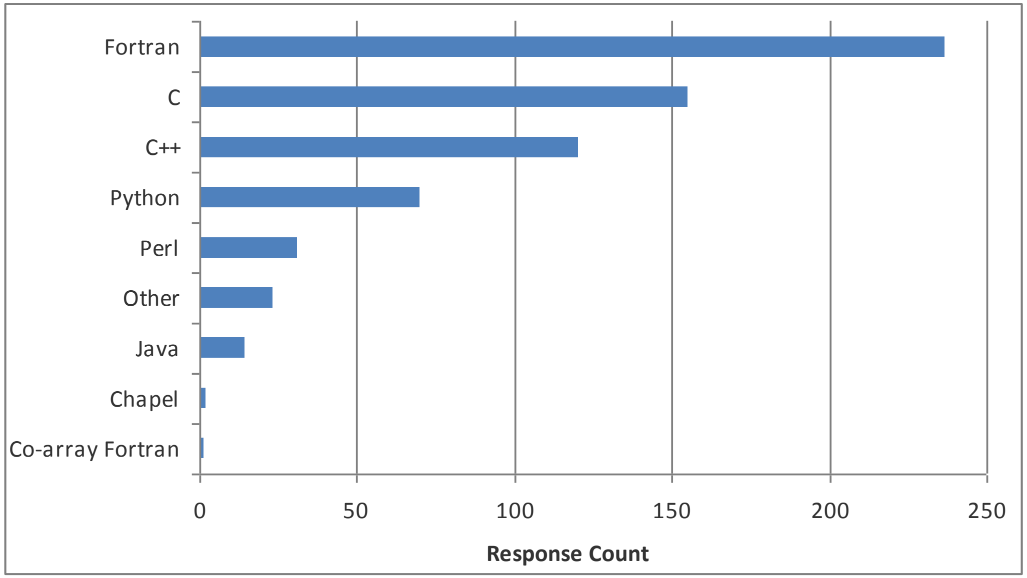
Parallel Programming Languages and Approaches

Contents

- A Little Bit of History
  - Non-Parallel Programming Languages
  - Early Parallel Languages
- Current Status of Parallel Programming
  - Parallelisation Strategies
  - Mainstream HPC
- Alternative Parallel Programming Languages
  - Single-Sided Communication
  - PGAS
- Final Remarks and Summary

Non-Parallel Programming Languages

- Serial languages important
  - General scientific computing
  - Basis for parallel languages
- PRACE Survey results:



- PRACE Survey indicates that nearly all applications are written in:
  - Fortran: well suited for scientific computing
  - C/C++: allows good access to hardware
- Supplemented by
  - Scripts using Python, PERL and BASH
  - PGAS languages starting to be used

Data Parallel

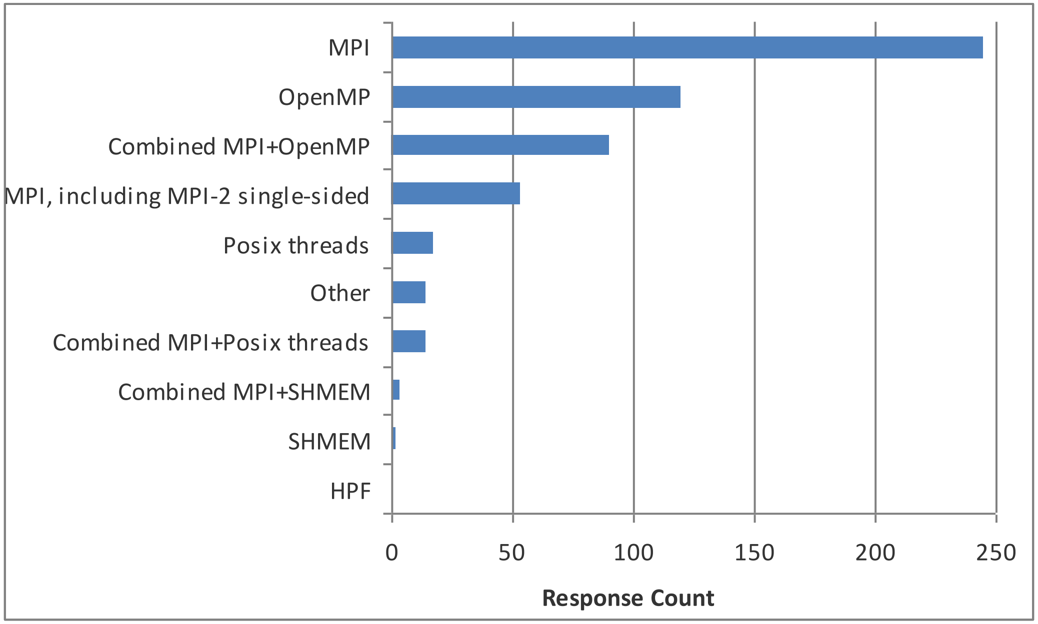
- *Processors perform similar operations across elements in an array*
- Higher level programming paradigm, characterised by:
  - single-threaded control
  - global name space
  - loosely synchronous processes
  - parallelism implied by operations applied to data
  - compiler directives
- Data parallel languages: generally serial language (e.g., F90) plus
  - compiler directives (e.g., for data distribution)
  - first class language constructs to support parallelism
  - new intrinsics and library functions
- Paradigm well suited to a number of early (SIMD) parallel computers
  - Connection Machine, DAP, MasPar,…

Data Parallel II

- Many data parallel languages implemented:
  - Fortran-Plus, DAP Fortran, MP Fortran, CM Fortran, *LISP, C*, CRAFT, Fortran D, Vienna Fortran
- Languages expressed data parallel operations differently
- Machine-specific languages meant poor portability
- Needed a portable standard: High Performance Fortran
- Easy to port codes to, but performance could rarely match that from message passing codes
  - Struggled to gain broad popularity

Parallelisation Strategies

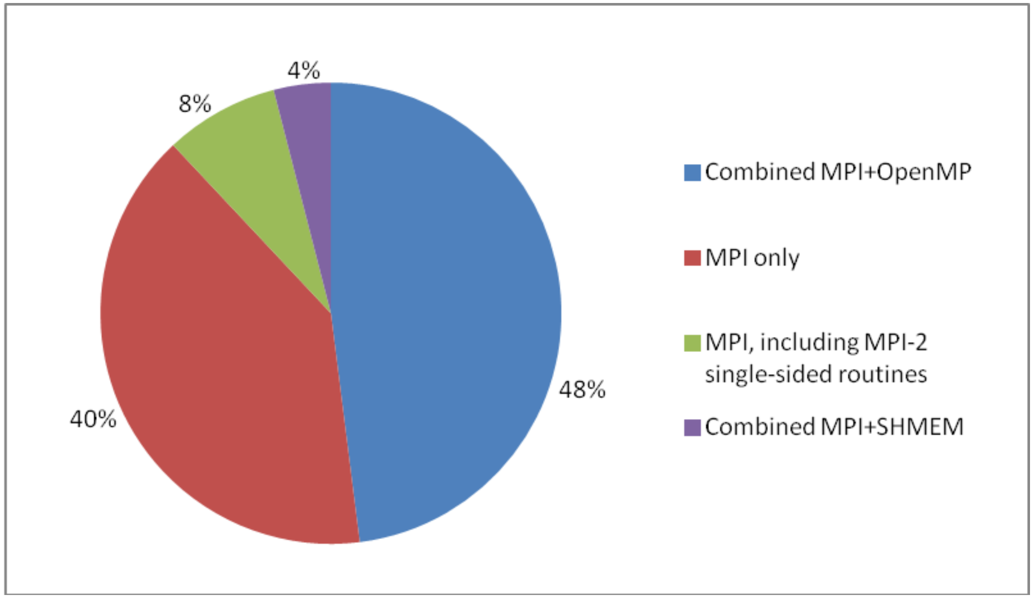
- PRACE asked more than 400 European HPC users
  - *"Which parallelisation implementations do you use?"*



- Unsurprisingly, most popular answers were MPI and/or OpenMP
  - Some users of Single-Sided communications

Parallelisation Strategies II

- PRACE also asked users of very largest systems:
  - *"Which parallelisation method does your application use?"*



- Most popular: "MPI Only" and "Combined MPI+OpenMP"
  - 12% used single-sided routines

Mainstream HPC

- For the last 15+years, most HPC cycles on large systems have been used to run MPI programs, written in Fortran or C/C++
  - Plus OpenMP used on shared memory systems/nodes

- However, there are now reasons why this may be changing:
  - Currently, HPC systems have increasingly large numbers of cores, but the individual core performance is relatively static
  - There are new challenges in exploiting future Exascale systems
- So, alongside mainstream HPC, there is also significant activity in:
  - Single-sided communication
  - PGAS languages
  - Accelerators
  - Hybrid approaches

Shared Memory

- *Multiple threads sharing global memory*
- Developed for systems with shared memory (MIMD-SM)
- Program loop iterations can be distributed to threads
  - Each thread can refer to private objects within a parallel context
- Implementation
  - Threads map to user threads running on one shared memory node
  - Extensions to distributed memory not so successful
- Posix Threads/PThreads is a portable standard for threading
- Vendors had various shared-memory directives
- OpenMP developed as common standard for HPC
  - OpenMP is a good model to use within a node
  - More recent task features

Message Passing

- *Processes cooperate to solve problem by exchanging data*

- Can be used on most architectures
  - Especially suited for distributed memory systems (MIMD-DM)
- The message passing model is based on the notion of *processes*
  - *Process*: an instance of a running program, together with its data
- Each process has access only to its own data
  - i.e., all variables are private
- Processes communicate by sending+receiving messages
  - Typically library calls from a conventional sequential language
- During the 1980s, an explosion in languages and libraries
  - CS Tools, OCCAM, CHIMP (developed by EPCC), PVM, PARMACS, …

MPI: Message Passing Interface

- *De facto* standard developed by working group of around 60 vendors and researchers from 40 organisations in USA and Europe
  - Took two years
  - MPI-1 released in 1993
  - Built on experiences from previous message passing libraries
- MPI's prime goals are:
  - To provide source-code portability
  - To allow efficient implementation
- MPI-2 was released in 1996
  - New features: parallel I/O, dynamic process management and remote memory operations (single-sided communication)

- Now, MPI is used by nearly all message-passing programs

Single-Sided Communication

- *Allows direct access to memory of other processors*
  - Process can access total memory, even on distributed memory systems
- Simpler protocol can bring performance benefits
  - But requires thinking about synchronisation, remote addresses, caching...

- Key routines
  - PUT is a remote write
  - GET is a remote read

- Libraries give PGAS functionality
- Vendor-specific libraries
  - SHMEM (Cray/SGI), LAPI (IBM)
- Portable implementations
  - MPI-2, OpenSHMEM

Single-Sided Communication

- Single-sided comms a major part of MPI-2 standard
  - Quite general and portable to most platforms
  - However, portability and robustness can have an impact on latency
  - Quite complicated and messy to use
- Better performance from lower-level interfaces e.g.SHMEM
  - Originally developed by Cray but a variety of similar implementations were developed on other platforms
  - Simple interface but hard to program correctly
- OpenSHMEM
  - New initiative to provide standard interface
  - See `http://www.openshmem.org`

Partitioned Global Address Space

- *Access to local memory via standard program mechanisms plus access to remote memory directly supported by language*

- The combination of access to all data plus also exploiting locality could give good performance and scaling
- Well suited to modern MIMD systems with multicore (shared memory) nodes
- Newly popular approach initially driven by US funding
  - Productive, Easy-to-use, Reliable Computing System (PERCS) project funded by DARPA's High Productivity Computing Systems (HPCS)

PGAS II

- Currently active and enthusiastic community
- Very wide variety of languages under the PGAS banner
  - See **`http://www.pgas.org`**
  - Including: CAF, UPC, Titanium, Fortress, X10, CAF 2.0, Chapel, Global Arrays, HPF?, …
- Often, these languages have more differences than similarities…

PGAS Languages

- Broad range of PGAS languages makes it difficult to choose
- Currently, CAF and UPC are probably most relevant
  - Cray's compilers and hardware now support CAF and UPC in quite an efficient manner
- CAF: Fortran with Coarrays
  - Minimal addition to Fortran to support parallelism
  - Incorporated in Fortran 2008 standard!
- UPC: Unified Parallel C
  - Adding parallel features to C

Why do Languages Survive or Die?

- It is not always entirely clear why some languages and approaches thrive while others fade away…
- However, languages which survive do have a number of common characteristics
  - Appropriate model for current hardware
  - Good portability
  - Ease of use
  - Applicable to a broad range of problems
  - Strong engagement from both vendors and user communities
  - Efficient implementations available

PGAS Libraries

- This course focuses on PGAS libraries – why?

- Language neutral
  - can program in either C or Fortran
- Does not require compiler functionality
  - greater portability between platforms
- PGAS languages often layered on single-sided libraries
  - learning library helps understanding of language characteristics

- Cray architectures have very good PGAS performance

Summary

- Development of portable standards have been essential for uptake of new parallel programming ideas
- Mainstream HPC is currently based on MPI and OpenMP
  - However, there are alternatives
- Exascale challenges have injected new life into novel parallel programming languages and approaches
- The remainder of this course focuses on PGAS libraries

References

- PRACE-PP
  - **D6.1: Identification and Categorisation of Applications and Initial Benchmarks Suite,** *Alan Simpson, Mark Bull and Jon Hill, EPCC*
- PRACE-1IP
  - **D7.4.1: Applications and user requirements for Tier-0 systems,** *Mark Bull (EPCC), Xu Guo (EPCC) and Ioannis Liabotis (GRNET)*
  - **D7.4.3: Tier-0 Applications and Systems Usage,** *Xu Guo and Mark Bull, EPCC*