Summary

• Background

• Introduction in algorithms and applications
  – Methodology to develop efficient parallel (distributed-memory) algorithms
  – Understand various forms of overhead (communication, load imbalance, search overhead, synchronization)
  – Understand various distributions (blockwise, cyclic)
  – Understand various load balancing strategies (static, dynamic master/worker model)
  – Understand correctness problems (e.g. message ordering)
Summary

• Parallel machines and architectures
  – Processor organizations, topologies, criteria
  – Types of parallel machines
    • arrays/vectors, shared-memory, distributed memory
  – Routing
  – Flynn’s taxonomy

  – What are cluster computers?
  – What networks do real machines (like the Blue Gene) use?

  – Speedup, efficiency (+ their implications), Amdahl’s law
Summary

- Programming methods, languages, and environments
  - Different forms of message passing
    - naming, explicit/implicit receive, synchronous/asynchronous sending
  - Select statement
  - SR primitives (not syntax)
  - MPI: message passing primitives, collective communication
    - Various communication modes
  - Java parallel programming model and primitives
  - HPF: problems with automatic parallelization; division of work between programmer and HPF compiler; alignment/distribution primitives; performance implications
Summary

• Applications
  – Search algorithm (TDS):
    • use asynchronous communication + clever (transposition-driven) scheduling
  – N-body problems:
    • load balancing and communication (locality) optimizations, costzones, performance comparison
  – Climate modeling:
    • Main problem is communication between CPU and GPU; solved by performance modeling
Summary and conclusions

- Higher performance cannot be reached by increasing clock frequencies anymore
- Solution: introduction of large-scale parallelism
  - Many-cores are here to stay
- Multiple cores on a chip
  - Today:
    - Up to 61 CPU cores in a node
    - Up to 3200 cores on a single GPU
  - Host system can contain multiple GPUs: 10,000+ cores
  - We can build clusters of these nodes!
- Future: 100,000s – millions of cores?
Summary and conclusions

- Many different types of many-core hardware

- Very different properties
  - Performance
  - Programmability

- The memory is the main bottleneck of all these platforms
  - Different memory spaces pose different challenges

- Performance analysis and estimation
  - Amdahl’s Law
  - Arithmetic intensity / Operational intensity
  - Roofline model
Many different many-core programming models

Most models are hardware-induced, require low-level optimizations:

- Vectorization
- Coalescing
- Explicit software caching (shared memory on GPU)

Future

- Cuda? OpenCL?
- OpenACC?