Introduction to High Performance Computing

A Blue Waters Online Course
Fall 2016

David Keyes, Instructor

Professor of Applied Mathematics and Computational Science
Director, Extreme Computing Research Center

King Abdullah University of Science and Technology
Parallel Algorithms

David E. Keyes
King Abdullah University of Science and Technology

with acknowledgments to
Thomas Sterling
University of Indiana, Bloomington
Four steps to create a parallel program

- Decomposition of computation in tasks
- Assignment of tasks to processes
- Orchestration of data access, communication & synchronization
- Mapping processes to processors

• First *three* involve the algorithm
• The *last* involves the architecture

\[\text{c/o D. E. Culler, UC Berkeley, et al.}\]
Co-design

- The one-way flow of the previous diagram is necessary, but usually *insufficient* if the goal is high performance
- Normally, we must *tune* the algorithm to the architecture
- This requires *measuring* the performance of the result and making changes to one or more stages of the algorithm
- A performance *model* should allow fewer iterations around the loop
Aside

• Occasionally, I will insert a stop sign to suggest that a student in self-learning mode – or an instructor in class mode – pause the video playback and brainstorm on what comes next, before proceeding

• Like having a performance model before collecting the data, anticipating what is coming next will help learn it – whether the anticipation proves met or not

• The video will not stop automatically and these signs can be ignored for fast review

• Probably, I will pause too little… so pause it, yourself!
What might you tune?

- Type of decomposition
- Number of tasks
- Clustering of tasks into processors
- Amount of information exchanged
- Frequency of exchanges
- Aggregation of exchanges
- Order of computation and communication
- Recomputation
- Mapping of tasks to processors
- Type of solution method
- Type of representation or basis
- Type of formulation (!)
- Goal of computation (!!)

None of these change the problem being solved, just how to solve it.

We may also be led to formulate a different problem that can be solved more efficiently!
Designing an HPC code – the diagram that launched SciDAC

c/o T. Dunning, SciDAC Report, 2000
“Code optimization” here refers to numerical algorithmics. Performance optimization is a different specialization.
Some important algorithms

- In 2000, Jack Dongarra & Francis Sullivan whimsically introduced in *IEEE Computational Science and Engineering* a set of the “Top 10” algorithms of the previous century
  - actually, from the 1940s through the 1980s
  - some numerical, some non-numerical, and some “mixed”
  - typically simulation or optimization algorithms, or algorithms that support these tasks

- In 2008, Xindong Wu & Vipin Kumar systematically performed a similar selection focusing on data mining, for *Knowledge and Information Systems*
  - half from the 1990’s and half earlier
  - typically statistical and inferential tools
More important algorithms

• In 2004, Phil Colella, in a talk entitled *Defining Software Requirements for Scientific Computing*, named a set of seven key algorithms the “Seven Dwarves”
  – giant in importance (!)

• In 2006, a large team including David Patterson, John Shalf, and Kathy Yelick wrote a report entitled *Parallel Computing Research: A View from Berkeley*, in which the original 7 dwarves were complemented by 6 more from discrete mathematics – 13 in all

• In 2009, a large team led by Vladimir Safanov at St. Petersburg, published open source implementations of all 13 dwarves under the *Parallel Dwarfs project*  
  – [https://paralleldwarfs.codeplex.com](https://paralleldwarfs.codeplex.com)

Could be a good course project to push one of these along
What would you put on these lists?
1946 — The Monte Carlo method.
1947 — Simplex Method for Linear Programming.
1950 — Krylov Subspace Iteration Method.
1951 — The Decompositional Approach to Matrix Computations.
1957 — The Fortran Compiler.
1959 — QR Algorithm for Computing Eigenvalues.
1965 — Fast Fourier Transform.
1977 — Integer Relation Detection.
1951 – $k$ Nearest Neighbor Classification ($k$NN)
1957 – $K$-means Clustering
1961 – Decision Trees
1977 – Expectation Maximization
1984 – Classification and Regression Trees (CART)
1994 – Apriori Algorithm
1995 – Support Vector Machines
1997 – Naïve Bayes
1997 – AdaBoost
1998 – PageRank
Rio Yokota’s interpretation of the seven dwarves

N-body | Structured Grids | Unstructured Grids | Fast Fourier Transform | Sparse Linear Algebra | Dense Linear Algebra

Dopey | Bashful | Sneezy | Sleepy | Happy | Grumpy | Doc

Heigh Ho...
The Scalable Hierarchical 5

- 1960’s – Fast Fourier Transform
- 1970’s – Multigrid Methods
- 1980’s – Fast Multipole Methods
- 1990’s – Sparse Grid Methods
- 2000’s – Hierarchically Low-Rank Matrix Methods

2016 KAUST Workshop on Scalable Hierarchical Algorithms for Extreme Computing (SHAXC'16)
High performance implementations

• All high performance implementations of these algorithms will exploit parallelism

• A goal of the course will be to classify parallel approaches and associate them with
  – Hardware that can support them
  – Algorithms that can benefit from them
  – Software that can implement them

• A successful student
  – builds categories
  – populates them with examples
  – connects new material to an expanding reference set

• To formulate parallel programming categories, we look back 50 years to …
Flynn's taxonomy (1966) of parallel architectures

Many significant scientific problems require the use of prodigious amounts of computing time. In order to handle these problems adequately, the large-scale scientific computer has been developed. This computer addresses itself to a class of problems characterized by having a high ratio of computing requirement to input/output requirements (a partially de facto situation caused by the unavailability of matching input/output equipment). The complexity of these processors, coupled with the advancement of the state of the computing art they represent has focused attention on scientific computers. Insight thus gained is frequently a predictor of computer developments on a more universal basis. This paper [reviews] possible organizations starting with “concurrent” organizations presently in operation and examining other theoretical possibilities.
Flynn's analysis (1972) of architectural effectiveness

The historic view of parallelism [...] is probably best represented by Amdahl [...] that certain operations [...] must be done in an absolutely sequential basis. These operations include, for example, the ordinary housekeeping operations in a program. In order to achieve any effectiveness at all [...] parallel organizations processing N streams must have substantially less than 1 / N x 100 percent of absolutely sequential instruction segments. One can then proceed to show that typically for large N this does not exist in conventional programs. A major difficulty with this analysis lies in the [implication...] that what exists today in the way of programming procedures and algorithms must also exist in the future.
Flynn’s hardware taxonomy

Single Program, Multiple Data (SPMD) is a natural generalization of Single Instruction, Multiple Data (SIMD) when each processing unit executes its own local copy of the instruction stream. These copies can branch differently depending upon the data.

C/o The Wikipedia
## 7 parallel algorithm paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embarrassingly Parallel</td>
<td>Monte Carlo</td>
</tr>
<tr>
<td>Fork-Join</td>
<td>OpenMP parallel for-loop</td>
</tr>
<tr>
<td>Manager-Worker</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>Divide-Conquer-Combine</td>
<td>Parallel Sort</td>
</tr>
<tr>
<td>Halo exchange</td>
<td>Stencil-based PDE solvers (FD, FV, FE)</td>
</tr>
<tr>
<td>Permutation</td>
<td>Matrix-matrix multiplication (Cannon’s algorithm)</td>
</tr>
<tr>
<td>Task Dataflow</td>
<td>Breadth-first Search</td>
</tr>
</tbody>
</table>

c/o Thomas Sterling
Embarrassingly Parallel

• A problem that requires little or no effort to identify and launch many concurrent tasks, and in which there is no internal synchronization or other communication or load balancing concern, is called “embarrassingly parallel”
  – The term dates to 1986, in an article by Cleve Moler (founder of MATLAB)
  – “Pleasingly parallel” is a popular alternative

• The classic such algorithm is Monte Carlo (1946), which is the first of the “Top 10” algorithms of the last century
  – Monte Carlo also has a popular alternative: the Metropolis Algorithm, named for Nicholas Metropolis who used it in early computations of the Ising Model of phase change in 1952
Monte Carlo

- Monte Carlo is a statistical approach for studying systems with a large number of degrees of freedom, such as neutron transport (for which it was first described by Von Neumann in 1947)
- It is the algorithm of choice for integration in high dimensions but it can be demonstrated in low, here two
Embarrassingly Parallel example:

Monte Carlo

- Monte Carlo is arbitrarily parallelizable, but agonizingly slow
  - convergence rate is proportional to $N^{-1/2}$
    - increasing work (here the number of samples) from 10 to 1,000 gives only a reduction of error of about a factor of 10
  - ideally, we would like convergence rate like $N^{-2}$
    - increasing work from 10 to 1,000 would reduce error by a factor of 10,000
    - we are accustomed much better rates in the quadrature of continuous functions

- In high dimensions, when measured in terms of function evaluations, Monte Carlo can be the method of choice, because of the curse of dimensionality, but beware of an indictment attributed to Von Neumann: “Anyone using Monte Carlo is in a state of sin.”
Monte Carlo advances

- Markov Chain Monte Carlo (MCMC) is a form of Monte Carlo that chooses smartly conditioned random samples
  - not statistically independent, but correlated for efficient “mixing”
  - includes: Metropolis-Hastings (1970), Gibbs sampling (1984), and many others that restore some respectability to MC 😊
- Multi-level Monte Carlo (MLMC) is a form of Monte Carlo that evaluates its samples on a hierarchy of models, coarsened from the model for which the solution is sought
  - most of the work is done on the coarser models, which are often exponentially cheaper, while obtaining the accuracy of the finest
  - invented by Mike Giles (2008) and very hot presently
- Of course, there is also MLMCMC, the methods MCMC and MLMC being composable
- Quantum Monte Carlo (QMC) is an application to quantum many-body systems – many computer codes
Fork-Join

- Launches parallel threads within an overall serial context
- Exploits concurrency independent tasks that become available in batches
- May incur overhead of creation and termination of the threads
- Presumes that the concurrent tasks require similar time to completion
Fork-Join example:

OpenMP loop

```
#pragma omp parallel private(p)
{
  p = 5;
  #pragma omp for
  for (i=0; i < 25; i++)
    a[i] = b[i] + p*(i+3);
}
```

- Code expresses 25 independent ops in for-loop
- OpenMP pragma directs the compiler to batch them concurrently
- Storage of a[] and b[] is shared – visible and writeable to all threads of execution
- For OpenMP standard (and training) see openmp.org
Aside

• This is not a course in parallel *programming*

• Acquisition of skill in parallel programming (using MPI, OpenMP, CUDA, etc.) is encouraged
  – short courses are available at many conferences, at many universities, and online
  – rich literature, excellent open source demos

• Programming produces insight and intuition hard to obtain from other investments of time

• Fortunately, the maturity of parallel computing today allows many computational scientists to be productive at a higher level
  – e.g., working within an API like PETSc, Trilinos, …
This paradigm is classically known as “master-slave”.
Manager assigns individual tasks to workers, receives the responses and dynamically constructs and assigns new tasks until work is complete.
Unlike Fork-Join, it does not presume that the tasks are well balanced, and it is not pre-scheduled.
Workers do not synchronize or exchange among selves.
Manager-Worker example:

Global parallel genetic algorithm

- Genetic algorithms are efficient stochastic search methods based on principles of natural selection and genetics
- They find an optimum by manipulating a population of candidate solutions against a fitness function
- Good candidates are preferentially selected to mate (combine their traits by crossover) and reproduce
- Many variations exist, including mutations outside of the parents
- The population can allow global exchanges or can be confined to different islands
Manager-Worker example:

Global parallel genetic algorithm

- Simple manager-worker

- Hierarchical model
Divide-Conquer-Combine

1. Given a “large” instance of a problem, partition it into two or more (independent) subproblems
2. Solve the subproblems (concurrently)
3. Merge the subsolutions into a solution of the original
   
   Recur if step 2 is still “large”
Divide-Conquer-Combine example:

**Quicksort**

- Named one of the “Top 10” algorithms of the last century
  - See introductory article by Joseph Jaja
- Invented in 1962 by Tony Hoare (born 1934, Turing Prize winner for definition and design of programming languages, long list of professional honors)
- Important not just as an early paradigm for parallel “divide-and-conquer”, but also as an example of a randomized algorithm
  - Randomized algorithms are extremely hot today, in both discrete and continuous forms
- Long history of complexity analysis for different cases of algorithm choices and input conditions
- Worst case running time is $O(N^2)$ for input size $N$, but with probability $1-N^{-c}$, it is $O(N \log N)$ – an “optimal” algorithm
Divide-Conquer-Combine example:

Quicksort

Given List of numbers \{ 3, 14, 15, 12, 9, 7, 5 \}  
Random choice of pivot: 12

Low list \{3,9,7,5\}  
Random choice of pivot: 7

\{3,5\}  
Random choice of pivot: 3

\{}  
\{5\}

High list \{14,15\}  
Random choice of pivot: 15

\{14\}

Concatenated result: \{3,5,7,9,12,14,15\}

The numbers represent a orderable set of keys in records to be sorted
Divide-Conquer-Combine example:

Parallel sample quicksort

As evenly as possible distribute $N$ keys among $P$ processors (here $N=8$ and $P=2$)

- Process 0: $\{3, 14, 15, 12\}$
- Process 1: $\{9, 7, 5, 10\}$

Run sequential quicksort on local data

- Process 0: $\{3, 12, 14, 15\}$
- Process 1: $\{5, 7, 9, 10\}$

Sample the sorted sublists at every $N/P^2$ locations

- Sample Set:
  - Process 0: $\{3, 14\}$
  - Process 1: $\{5, 9\}$
Divide-Conquer-Combine example:

Parallel sample quicksort

Gather samples to the root ($P=0$) and run sequential quicksort

Broadcast $P-1$ pivot values to all processes

Divide local sorted segments on each process into $P$ segments based on pivots
Divide-Conquer-Combine example:

**Parallel sample quicksort**

Perform an all-to-all communication on the $P$ segments, with $P_i$ keeping the $i^{th}$ segment and sending the $j^{th}$ segment to $P_j$

![Diagram showing communication and sorting steps]

Merge incoming (already sorted) sub-segments into local lists

- **Process 0**
  - Pivot: 9
  - Initial segments: $\{3\}$, $\{12,14,15\}$
  - Sorted segments: $\{3,5,7\}$
  - Process 0

- **Process 1**
  - Pivot: 9
  - Initial segments: $\{5,7\}$, $\{10\}$
  - Sorted segments: $\{10,12,14,15\}$
  - Process 1

**Final result:** $\{3,5,7,9,10,12,14,15\}$

The final list of keys is distributed, which is probably what is desired if $N$ is large compared to what a single processor can store in the first place.

The rest of the record volume associated with each sorted key can be moved into place.
Divide-Conquer-Combine example:

Quicksort

• After 54 years, quicksort remains the sorting routine of choice for large $N$ unless there exists additional detailed information about the input

• Other means of selecting the pivot exist
  – Clearly selecting the median is best
  – Normally, there is no fast way to find the median
  – Fixed position works poorly unless the input is random
  – Finding the median of a randomly chosen subset may be superior to a single random choice

• Complexity can be measured in individual comparison operations, or can incorporate memory and communication costs
Halo Exchange

• Many computations, especially partial differential equations, are executed by SPMD programs in which each process advances at least one subdomain of the overall PDE domain, and every subdomain is assigned to at least one process (usually a 1-to-1 mapping)

• By their mathematical nature, PDE codes discretize derivatives, which are approximated with local “stencils”

• Partitioning into subdomains requires replication of the points of a stencil that belong to a neighboring subdomains, the replicated region called a “halo”

• Parallel implementation requires (usually symmetrical) nearest-neighbor communication
Halo Exchange example:

2\textsuperscript{nd}-order Laplacian in 2D

- Halo exchange is regular and frequent and is built into communication subroutines, e.g., in the standard Message-Passing Interface, MPI

owned cell
halo exchange cell
ghost boundary cell
unused for 5-pt star

8x8 domain in four 4x4 subdomains, with halos and ghosts

c/o Fabian Dournac
Halo Exchange example:

4th-order Laplacian in 2D

- For higher-order discretizations, stencils and halos need to be wider.

- Everything in the black square is owned by one interior subdomain (with no boundary points).

- The magenta is a double-wide halo.

c/o Michael Flynn
Halo Exchange example:

Sparse matrix-vector multiplication

- Stencil evaluation is a special, structured form of a more general operation, sparse matrix-vector multiplication
- For a matrix of size $N \times N$ and vector of size $N$, matrix-vector multiplication is given by

$$x_i = \sum_{j=0}^{N-1} A_{ij} b_j$$

where $A_{ij}$ is the $(i,j)^{th}$ element of the matrix and $b_j$ is the $j^{th}$ element of the vector
- For a sparse matrix, however, most of the $A_{ij}$ values are zero, suggesting that memory should be allocated for every element
  - for 2nd-order Laplacian, $A_{ii} = 4$ and just four off-diagonals are nonzero
Halo Exchange example:

**Sparse matrix-vector multiplication**

- Sparse matrix $A_{ij}$ can use compressed sparse row (CSR) format
- Then the matrix, the input vector $b$, and the output vector $x$ can all be partitioned by rows with a block of rows assigned to each processor
- Information from vector $b$ is then potentially nonlocal to the processors computing a local pieces of $x$
Halo Exchange example:

Sparse matrix-vector multiplication

\[ x_i = \sum_{j=0}^{n_i} A_{ij} b_j = \sum_{\text{local}} A_{ij} b_j + \sum_{\text{nonlocal}} A_{ij} b_j \]
Permutation

- Permutation is a class of data-parallel SPMD algorithms, in which the same local program is simultaneously applied to different data in a series of stages.
- Between each stage is an all-to-all permutation of the data.
- The Fast Fourier Transform is a famous example, considered later.
- Continuing upon matrix-vector multiplication, we consider matrix-matrix multiplication, \( C = A \times B \).
- Cannon’s algorithm (1999) is a special form of dense matrix-matrix multiplication in which two \( N \times N \) matrices are multiplied by a \( \sqrt{P} \times \sqrt{P} \) array of processors, each of which owns a \( \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} \) block of \( A, B \), and \( C \).
Permutation example:

**Dense matrix-matrix multiplication**

- Matrix-matrix multiplication is an $O(N^3)$ operation on $O(N^2)$ data

\[
C_{ij} = \sum_{k=0}^{N-1} A_{ik} B_{kj}
\]

- Its high arithmetic intensity (ratio of flops to bytes) allows it to “cover” data motion well

\[\begin{array}{cccc}
C_{00} & C_{01} & C_{02} & C_{03} \\
C_{10} & C_{11} & C_{12} & C_{13} \\
C_{20} & C_{21} & C_{22} & C_{23} \\
C_{30} & C_{31} & C_{32} & C_{33}
\end{array}\quad = \quad \begin{array}{cccc}
A_{00} & A_{01} & A_{02} & A_{03} \\
A_{10} & A_{11} & A_{12} & A_{13} \\
A_{20} & A_{21} & A_{22} & A_{23} \\
A_{30} & A_{31} & A_{32} & A_{33}
\end{array}\quad \begin{array}{cccc}
B_{00} & B_{01} & B_{02} & B_{03} \\
B_{10} & B_{11} & B_{12} & B_{13} \\
B_{20} & B_{21} & B_{22} & B_{23} \\
B_{30} & B_{31} & B_{32} & B_{33}
\end{array}\]
Permutation example:

**Dense matrix-matrix multiplication**

- Each block of the product on each processor is the result of $\sqrt{P}$ blocks of each of the factors

\[ C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} \]

- All but one of these pairs are initially nonlocal
Permutation example:

Dense matrix-matrix multiplication

- From the initial state shown

- Cannon’s algorithm is

  row $i$ of matrix $A$ is circularly shifted by $i$ blocks left
  col $j$ of matrix $B$ is circularly shifted by $j$ blocks up

  For $k = 0$ to $\sqrt{P} - 1$:

  $P_{ij}$ multiplies its two entries and accumulates
  each row of $A$ is circularly shifted 1 element left
  each col of $B$ is circularly shifted 1 element up
Permutation example:

**Dense matrix-matrix multiplication**

row $i$ of matrix $A$ is circularly shifted by $i$ blocks left
col $j$ of matrix $B$ is circularly shifted by $j$ blocks up

Cannon’s Algorithm: Setup
Permutation example:

**Dense matrix-matrix multiplication**

*Layout after initialization*

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$C_{00}$</td>
<td>$C_{01}$</td>
<td>$C_{02}$</td>
<td>$C_{03}$</td>
</tr>
<tr>
<td></td>
<td>$A_{00}$</td>
<td>$A_{01}$</td>
<td>$A_{02}$</td>
<td>$A_{03}$</td>
</tr>
<tr>
<td></td>
<td>$B_{00}$</td>
<td>$B_{11}$</td>
<td>$B_{22}$</td>
<td>$B_{33}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>9</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C_{10}$</td>
<td>$C_{11}$</td>
<td>$C_{12}$</td>
<td>$C_{13}$</td>
</tr>
<tr>
<td></td>
<td>$A_{11}$</td>
<td>$A_{12}$</td>
<td>$A_{13}$</td>
<td>$A_{10}$</td>
</tr>
<tr>
<td></td>
<td>$B_{10}$</td>
<td>$B_{21}$</td>
<td>$B_{32}$</td>
<td>$B_{03}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>6</th>
<th>10</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$C_{20}$</td>
<td>$C_{21}$</td>
<td>$C_{22}$</td>
<td>$C_{23}$</td>
</tr>
<tr>
<td></td>
<td>$A_{22}$</td>
<td>$A_{23}$</td>
<td>$A_{20}$</td>
<td>$A_{21}$</td>
</tr>
<tr>
<td></td>
<td>$B_{20}$</td>
<td>$B_{31}$</td>
<td>$B_{02}$</td>
<td>$B_{13}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>7</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$C_{30}$</td>
<td>$C_{31}$</td>
<td>$C_{32}$</td>
<td>$C_{33}$</td>
</tr>
<tr>
<td></td>
<td>$A_{33}$</td>
<td>$A_{30}$</td>
<td>$A_{31}$</td>
<td>$A_{32}$</td>
</tr>
<tr>
<td></td>
<td>$B_{30}$</td>
<td>$B_{01}$</td>
<td>$B_{12}$</td>
<td>$B_{23}$</td>
</tr>
</tbody>
</table>
Permutation example:

Dense matrix-matrix multiplication

Accumulate* and shift

\[ C_{ij} + = A_{i, [i+j+k]} B_{[i+j+k], j} \]

*The bracket \([i+j+k]\) is interpreted in the sense of modulus \(\sqrt{P}\) (4 in this example)
Permutation example:

Dense matrix-matrix multiplication

- The distribution after each phase is depicted here
- It is readily verified that the correct accumulations have occurred
Permutation example:

Dense matrix-matrix multiplication

Flow chart and pseudocode

\[ C_{ij} \leftarrow A_{i,[i+j+k]}B_{[i+j+k],j} \]
Task Dataflow

• While any parallel algorithm can be expressed as a directed graph where the nodes are tasks and the edges are data dependencies, many algorithms that explore graphs are themselves naturally expressed as task dataflow to maximize concurrency.

• A goal is to maximize the number of tasks that can be executed concurrently (breadth) and to minimize the critical path of dependences (depth).
Task Dataflow example:

**Breadth-first search**

- Breadth-first search is a key component of numerous larger programs and is tested in the Graph500 benchmark.
- A particular vertex is named as root.
- Each adjacent vertex to the root is then traversed first.
- When no more immediate root neighbors exist, previously labeled neighbors traverse their neighbors, thereby establishing the level (or distance) of every vertex from the root.
- It should be compared to a depth-first search, which explores as far as possible before backtracking.
Task Dataflow example:

**Breadth-first search**

Vertex labeling from a breadth-first search

Vertex labeling from a depth-first search

C/o Wikipedia
Task Dataflow example:

**Breadth-first search**

- Example for breadth-first search showing final level 0, level 1, and level 2
- Seven edges to traverse, on two of which the vertex has been previously visited, in sequential mode

Starting at root 8: 8, 4, 5, 1, 7, 9
Task Dataflow example:

**Breadth-first search**

- Vertices are partitioned by process each with its own edge list, including the process number of the adjacent vertex.
Task Dataflow example:

**Breadth-first search**

- To each vertex is associated a parent vertex label and a binary flag indicating if the vertex has been visited
- Parallel BFS is initialized
Task Dataflow example:

Breadth-first search

- At the first stage, a level 1 vertex is found on process 1, which then begins to search concurrently.
Task Dataflow example:

**Breadth-first search**

- Two stages are completed in the time of five edge traversals rather than seven for sequential
Summary

• Four steps in creating a parallel program
• Co-design of solution algorithm and problem formulation with architecture, to tune for performance
• Top 10 algorithms for simulation
• Top 10 algorithms for data mining
• 7 floating point “dwarves” and 6 discrete “dwarves”
• 5 optimal scalable, hierarchical algorithms
• Flynn’s classification for programming paradigms supported in the hardware
Summary

• Embarrassing Parallelism
  – Monte Carlo

• Fork-Join
  – OpenMP “for” loop

• Manager-Worker
  – Global Parallel Genetic Algorithm

• Divide-Conquer-Combine
  – Quicksort
Summary

• Halo Exchange
  – Stencil Evaluation / Sparse Matrix-Vector Multiplication

• Permutation
  – Dense Matrix-Matrix Multiplication

• Task Dataflow
  – Breadth-first Search
Principal slide credits

- Thomas Sterling (U Indiana)
- Matthew Anderson (U Indiana)
- Maciej Brodowicz (U Indiana)
- plus individual slides as marked