

# "An Off-The-Wall, Possibly CHARMING View of Future Parallel Application Development

Jim Browne

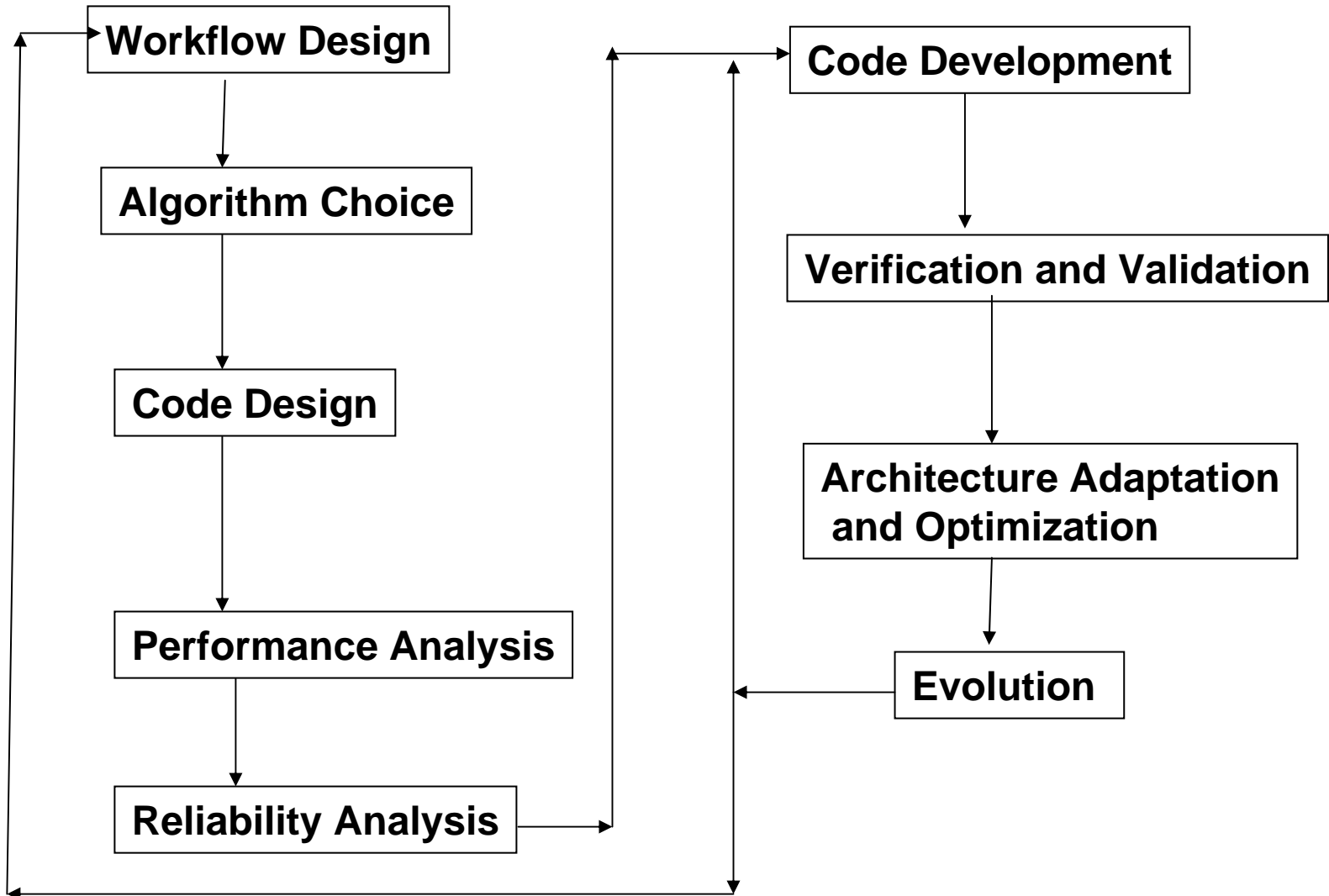
University of Texas

[browne@cs.utexas.edu](mailto:browne@cs.utexas.edu)

# "An Off-The-Wall, Possibly CHARMing View of Future Parallel Application Development"

Development methods for HPC applications change slowly and will continue to change slowly. It is thus safe to suggest radical changes because the chance they will be adopted quickly is low. This talk will sketch a few possible futures for HPC application development which are considerably different from current practice. The first part of the talk will sketch possible influences of development practices and the second some responses to these influences including, components, self-management, a merger of grid and HPC developments, tools based on expert systems technology.

# Application Development Process



# Future

- More complex execution environments
- Fundamental assumptions of underlying machine reliability may erode
- More complex system-oriented applications spanning multiple disciplines and time scales

What do these trends mean for application development tools?

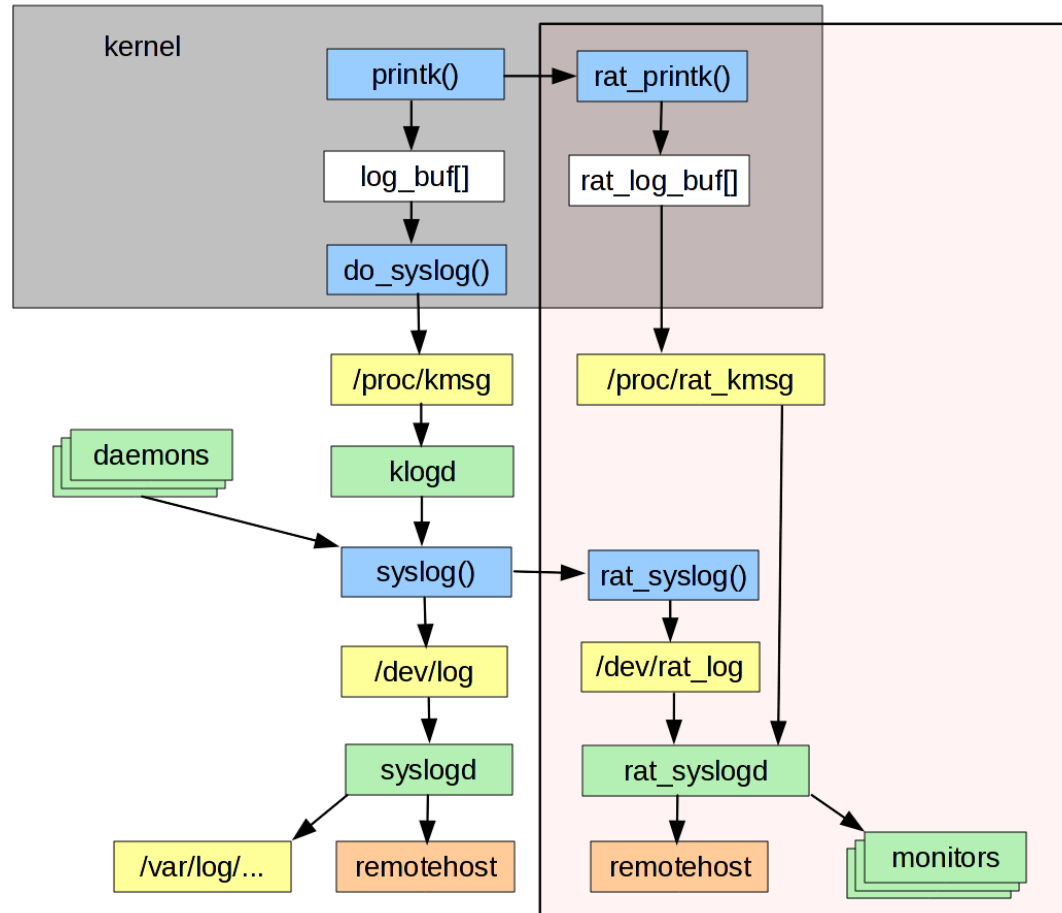
# What do these trends mean for application development tools?

- Greater breadth and depth of expert knowledge is needed
- Complexity of each step in the development process and thus complexity of tools will increase
- Tools may have to span multiple layers of software stack.
- User need (and demand?) for effective support will increase

# Example: Core/Chip/Node Parallelism

- Core/Chip/Node Architecture
  - Chips with  $2^N$  cores,  $N = 2, 4, 8, \dots$ 
    - Each core has memory hierarchy
  - Nodes with  $2^M$  chips,  $M = 2, 4, 8, \dots$ 
    - Nodes may be asymmetric and heterogeneous
  - Processes/Threads per node =  $K * N * M$
- Performance Limitations
  - Performance at core level – Resource optimization
  - Performance at chip level – Resource optimization + thread management
  - Performance at node level – thread management

# Example: Monitoring for Reliability



## Printk/syslog message logging rationalization

# Ongoing Studies

- NSF Reports
  - <http://www.nsf.gov/pubs/2007/nsf0728/index.jsp?org=NSF>
- Exascale
  - [http://www.exascale.org/iesp/Main\\_Page](http://www.exascale.org/iesp/Main_Page)
- RelXLayer
  - <http://www.relxlayer.org/>



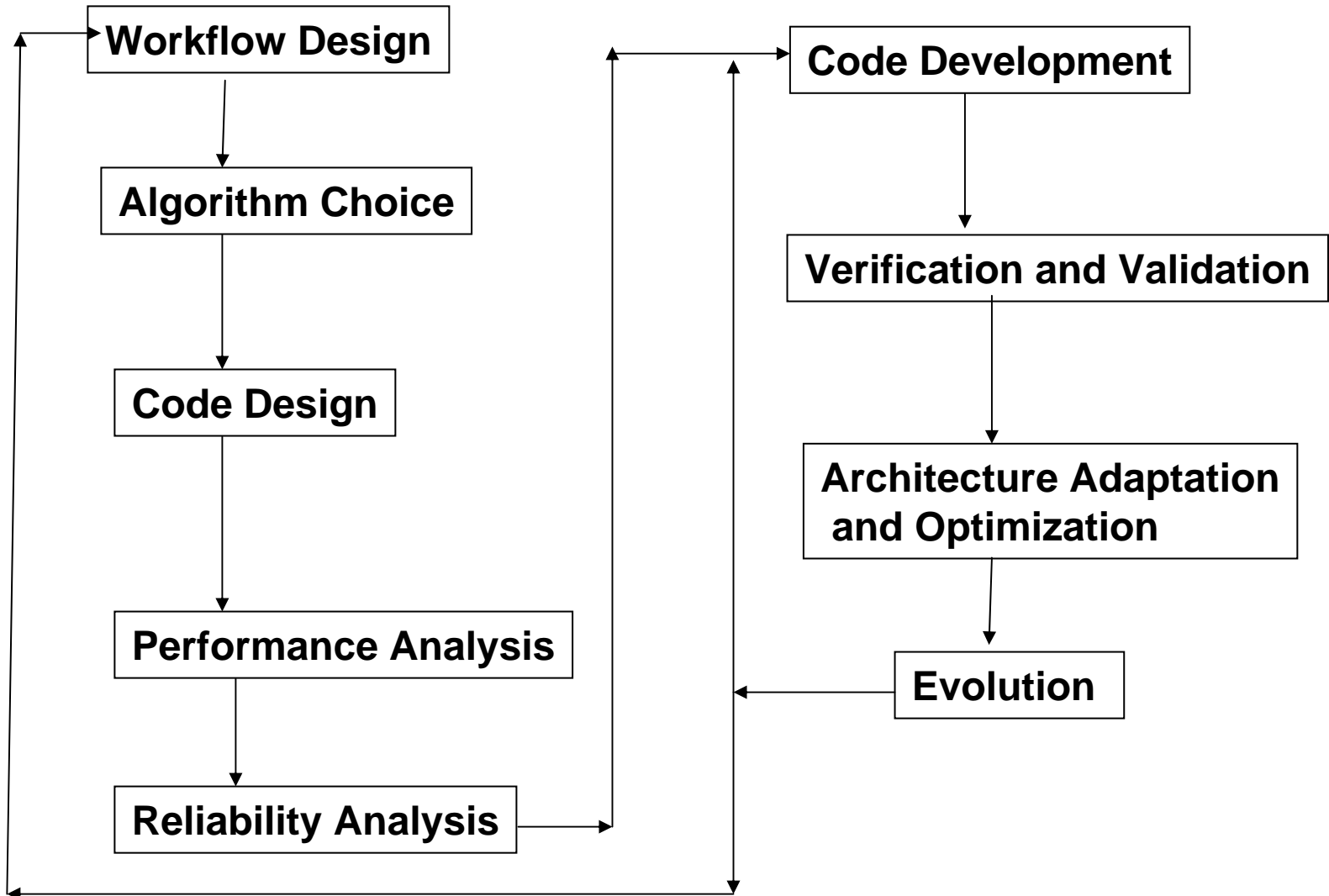
# Fundamental Principles for Tools

- Facility Capture of User Knowledge
- Guide Users to Desirable Choices
  - Interactive application of expert knowledge
- Automate Common (and Complex) Tasks
  - Automation Based on Expert Knowledge
    - Algorithm choice
    - Architectural adaptation and performance evaluation
    - .....

# Personal View

- Tools should guide users to (hierarchical) components with unitary functionality, controlled interfaces and simple control structures
- Why?
  - Facilitate each step of development
    - Testing and verification of correctness
    - Performance optimization
  - .....

# Application Development Process



# Tool Support – State of the Art

- Each Stage/Phase – Typically supported by different tools which don't communicate or interact.
- Some tools difficult to use and require expert knowledge.
- Fundamental principles enabling effective development not always followed by tool developers.

# Tool Support – State of the Art

- Each Stage/Phase supported by different tool which don't communicate or interact.
- Some tools difficult to use and require expert knowledge.
- Fundamental principles enabling effective development not always followed by tool developers.
- **Of Course, NONE OF THESE NASTY THINGS ARE TRUE of CHARM TOOL SET!**

# Tool Support – State of the Art

- Each Stage/Phase – Typically supported by different tools which don't communicate or interact.
- Some tools difficult to use and require expert knowledge.
- Fundamental principles enabling effective development not always followed by tool developers.

What should we (tool developers) be doing to meet future challenges?

# End to End Lifecycle Support

- Seamless end to end lifecycle support (customizable by application domain)
- Charm++ supports (to some degree):
  - Code Design
  - Reliability
  - Performance Analysis
  - Implementation
  - Verification
  - Architectural adaptation and optimization
  - .....

# Future for Tool Builders

- Increasing complexity of execution environments and applications will make it exceedingly difficult for a single tool developer (except maybe IBM with OPM) to develop comprehensive tool chains
  - Many domains of specialized knowledge will be required
  - Tool complexity will rise and managability will be an issue



# How To Develop Comprehensive Development Environments?

- Collaborative across university and open source groups will be needed.
- How to integrate tools?
  - Framework for Tool Integration?
  - Development of standard interfaces and interaction protocols for tools?
  - Form development consortia?
  - Have integration a goal for tool builders?

# Eclipse Framework

- How many use Eclipse?
- How many think Eclipse solves all these problems?
- How many think Eclipse is a potential framework for solutions to all these problems?

# Integration of Specialized Tools with Development Environments

- .Integratable tools must have very simple user interfaces
- Must work directly with outputs of development environment
- Must add significant value
- Example – Integrate PerfExpert: An automatic architectural adaptation and optimization with Charm++

# Intracore, Intrachip, Intranode Optimization for Charm++

- Charm++ : Asynchrony, thread management, communication optimization, load balancing
- PerfExpert: Intrachip, intranode resource optimization,
- Charm++ + PerfExpert
- PerfExpert built for integration with development environments

# PerfExpert

- Tool for architectural adaptation and performance optimization for multicore chips and multichip architectures
- Automates most of intra-node performance optimization for multicore chips and multichip nodes of large clusters.

# Project Goal

- Automate detection and characterization of performance bottlenecks
  - At core, chip, and node level
- Suggest optimizations for each bottleneck
  - Including code examples and compiler switches
  - Future: apply suggestions automatically
- Simplicity is paramount
  - Trivial user interface
  - Easily understandable output



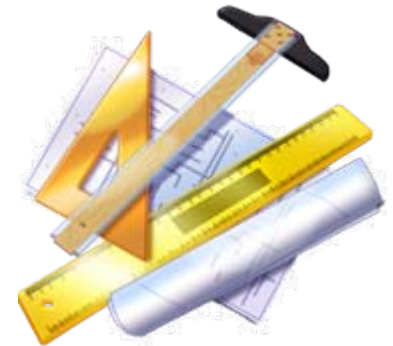
# PerfExpert Approach

- Gather performance counter measurements
  - Multiple runs with HPCToolkit
  - Sampling-based results for procedures and loops
- Combine results
  - Check variability, runtime, consistency, and integrity
- Compute and output assessment
  - Only for most important code sections
  - Correlate results from different thread counts



# PerfExpert v1.0

- Current features
  - Automatic bottleneck detection (uses HPCToolkit)
  - Extensive list of suggested optimizations with examples
  - Simple and intuitive interface
- Current capabilities
  - Based on general PAPI performance counters
  - Intra-node performance
  - Focus is on Ranger
- Single command line execution





# Bottleneck Identification and Characterization

- Diagnoses for potential bottlenecks
- FP, Branch, L1 cache, L2 cache, L3 cache, TLB, DRAM
- Automatically chooses measurements, does code executions, accumulates measurements, runs expert system with architecture specific rules and parameters to identify and **characterize** bottlenecks at procedure and loop nest levels.

# PerfExpert Performance Metric

- Cycles Per Instruction (CPI)
  - Compute upper bounds on CPI contribution for various groups (memory and TLB accesses, instruction groups)
    - $(BR\_INS * BR\_lat + BR\_MSP * BR\_miss\_lat) / TOT\_INS$
    - $(L1\_DCA * L1\_dlat + L2\_DCA * L2\_lat + L2\_DCM * Mem\_lat) / TOT\_INS$
- Benefits
  - Highlights key aspects and hides misleading details
  - Relative metric (less susceptible to non-determinism)

# Optimizations

- Suggests possible code optimizations depending on characterization of bottleneck.
- May offer specific code structures in some cases.
- Will suggest compiler switches for particular procedures.

# PerfExpert Output

```
PerfExpert v0.9
```

```
Copyright (c) 2009, The University of Texas at Austin. All rights reserved.
```

```
usage: PerfExpert.perl input [input]
```

```
total runtime of mmm1.csv is 0.03 seconds
```

```
matrixproduct (53.5% of the total runtime)
```

```
-----  
-  
WARNING: The runtime is too short to gather meaningful measurements.
```

```
cvtas_t_to_a (22.6% of the total runtime)
```

```
-----  
-  
WARNING: The cycle count variation is 33.3%, making the results unreliable.
```

```
WARNING: The runtime is too short to gather meaningful measurements.
```





# Suggestions with Examples

## If floating-point instructions are a problem

- **Reduce the number of floating-point instructions**
  - a) eliminate floating-point operations through distributivity  
 $d[i] = a[i] * b[i] + a[i] * c[i]; \rightarrow d[i] = a[i] * (b[i] + c[i]);$
- **Avoid divides**
  - b) compute the reciprocal once outside of loop and use multiplication inside the loop  
 $\text{loop } i \{a[i] = b[i] / c;\} \rightarrow \text{cinv} = 1.0 / c; \text{loop } i \{a[i] = b[i] * \text{cinv};\}$
- **Avoid square roots**
  - c) compare squared values instead of computing the square root  
 $\text{if } (x < \text{sqrt}(y)) \{\} \rightarrow \text{if } ((x < 0.0) || (x*x < y)) \{\}$
- **Speed up divide and square-root operations**
  - d) use float instead of double data type if loss of precision is acceptable  
 $\text{double } a[n]; \rightarrow \text{float } a[n];$
  - e) allow the compiler to trade off precision for speed  
try the “-prec-div”, “-prec-sqrt”, and “-pc32” compiler flags

# MangII Optimization Case Study

## If data accesses are a problem

- **Reduce the number of memory accesses**
  - a) copy data into local scalar variables and operate on the local copies
  - b) recompute values rather than loading them if doable with few operations
  - c) vectorize the code
- **Improve the data locality**
  - d) componentize important loops by factoring them into their own subroutines
  - e) employ loop blocking and interchange (change the order of the memory accesses)
  - f) reduce the number of memory areas (e.g., arrays) accessed simultaneously
  - g) split structs into hot and cold parts, where the hot part has a pointer to the cold part
- **Other**
  - h) use smaller types (e.g., float instead of double or short instead of int)
  - i) for small elements, allocate an array of elements instead of each element individually
  - j) align data, especially arrays and structs
  - k) pad memory areas so that temporal elements do not map to the same set in the cache



# Eliminate Inapplicable Suggestions

## If data accesses are a problem

- **Reduce the number of memory accesses**

- ~~a) copy data into local scalar variables and operate on the local copies~~
- ~~b) recompute values rather than loading them if doable with few operations~~
- c) vectorize the code

- **Improve the data locality**

- d) componentize important loops by factoring them into their own subroutines
- ~~e) employ loop blocking and interchange (change the order of the memory accesses)~~
- f) reduce the number of memory areas (e.g., arrays) accessed simultaneously
- ~~g) split structs into hot and cold parts, where the hot part has a pointer to the cold part~~

- **Other**

- h) use smaller types (e.g., float instead of double or short instead of int)
- ~~i) for small elements, allocate an array of elements instead of each element individually~~
- j) align data, especially arrays and structs
- k) pad memory areas so that temporal elements do not map to the same set in the cache



# Try Remaining Suggestions

## If data accesses are a problem



- **Reduce the number of memory accesses**

- ~~a) copy data into local scalar variables and operate on the local copies~~
- ~~b) recompute values rather than loading them if doable with few operations~~
- c) vectorize the code

- **Improve the data locality** 

- d) componentize important loops by factoring them into their own subroutines
- ~~e) employ loop blocking and interchange (change the order of the memory accesses)~~ 
- f) reduce the number of memory areas (e.g., arrays) accessed simultaneously
- ~~g) split structs into hot and cold parts, where the hot part has a pointer to the cold part~~ 

- **Other**

- h) use smaller types (e.g., float instead of double or short instead of int)
- ~~i) for small elements, allocate an array of elements instead of each element individually~~ 
- j) align data, especially arrays and structs
- k) pad memory areas so that temporal elements do not map to the same set in the cache 



# Related Work

- Automatic bottleneck analysis and remediation
  - PERCS project at IBM Research
    - Less automation for bottleneck identification and analysis
    - Not open source
  - PERI Autotuning project
  - Parallel Performance Wizard
    - Event trace analysis, program instrumentation
- Analysis tools with automated diagnosis
- Projects that target multicore optimizations

# Conclusions

- Charm++ and its off-shoots and out-growths comprise the most comprehensive parallel development environment in existence and has mostly been developed by one laboratory
- The complexity of future execution environments and applications will make it difficult to continue “going it alone” even for Charm++

# Conclusions - Continued

- There are many complementary tool development efforts.
- University and open source tool builders need to develop mechanisms for collaboration to develop coordinated, comprehensive lifecycle coverage tools for future systems and applications
- Charm++ is a natural leader for such an effort.