

# Techniques for Developing Distributed High Performance Computing Applications

---

Purushotham (Puri) Bangalore  
Dept. of Computer and Information Sciences  
University of Alabama at Birmingham

Distributed Programming Abstractions Workshop at the 2008 Mardi Gras Conference  
January 30, 2008, Baton Rouge, Louisiana.

# Outline

- Distributed HPC applications
- Issues in developing Grid applications
- Techniques for developing Grid applications
- Initial experiments and results
- Conclusions

# Types of Grid Applications

- Sequential/Batch jobs
- Embarrassingly parallel applications
- Traditional parallel applications - MPI, OpenMP, Pthreads, MPI-OpenMP hybrid
- Loosely coupled parallel applications
- Tightly coupled parallel applications
- Workflows - combination of above options
- Adaptive applications
- Traditional distributed programs and web/grid services

# Development vs. Deployment

- Development
  - The process of design, implementation, and testing an application
  - Could be performed on a local resource
- Deployment
  - Preparing for execution of an application on a specific computational platform
  - Includes migrating/porting and testing the application on the select platform
  - Grid environment used for better capability and/or capacity requirements

# Deploying Grid Applications

- Often referred to as Grid-enabling applications
- Adopting/wrapping an existing application to execute in the Grid
- Moving from execution on single resource to the Grid environment - this transition does not happen automatically
- Migrating and testing the application across multiple heterogeneous resources
- Distributing task/data across heterogeneous resources introduces interesting problems

# Deployment Options

- Manual job submission using command-line clients
- Create scripts from command-line tools
- Write a client using CoG toolkit or GAT
- Use a component framework
- Wrap applications as Grid services

# Sequential and Embarrassingly Parallel Applications

- Development: No major issues
- Deployment:
  - Application need to be tested on different resources to ensure that appropriate libraries and other runtime environment requirements are satisfied
  - Job resubmit and restart on partial failure need to be handled
  - A design template can be used to specify the computation and handle other deployment issues
  - Need to handle distribution of tasks across multiple resources
- Applications: BLAST and R

# Traditional Parallel Applications

- Development
  - Using one of the well-known parallel programming paradigms
  - Libraries, component frameworks, design templates, etc. could help development of these applications
- Deployment
  - Application performance profiling to select suitable application parameters
  - If application supports checkpointing, application can be checkpointed and migrated when better resources become available (requires application performance profile on different resources)
- Applications: NAMD and CBIR

# Common Issues

- Support for dynamic load balancing due to heterogeneity of resources
- Support to recover from failures/partial failures
- Support for job migration
- Support for application scheduling based on application characteristics (e.g., performance on specific resource, algorithm, input parameters, etc.)

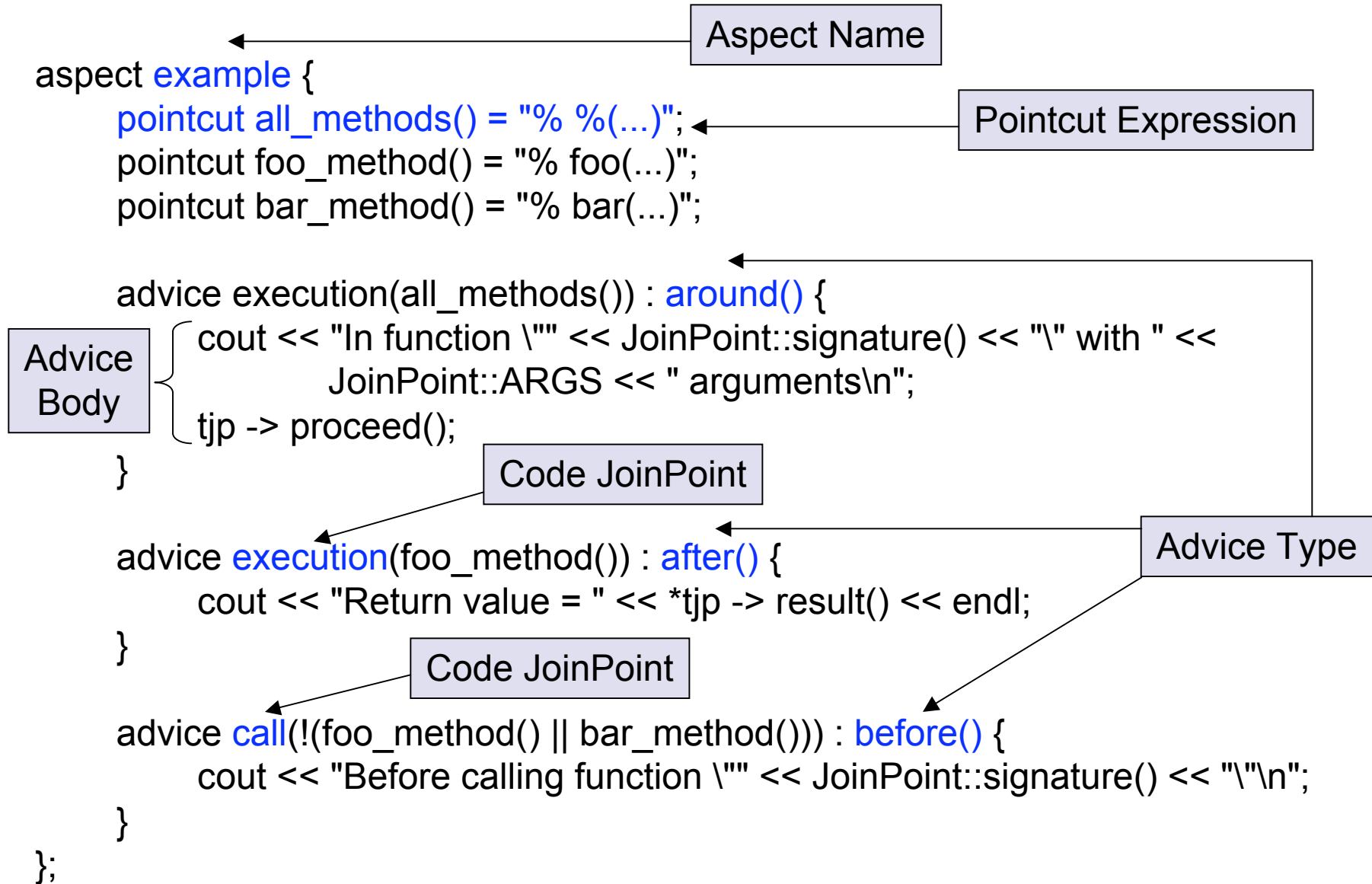
# Experiences Using Aspect Oriented Programming

---

# What is AOP?

- Aspect Oriented Programming (AOP) - modular programming paradigm that helps the programmer in separating cross-cutting concerns (*i.e.*, the concerns that cut through many modules)
- The program is modularized in such a way that the coupling between the modules is at its minima *i.e.*, each module performs a distinct task
- The existing program is not modified and desired functionality can be inserted or removed from an existing parallel application as needed

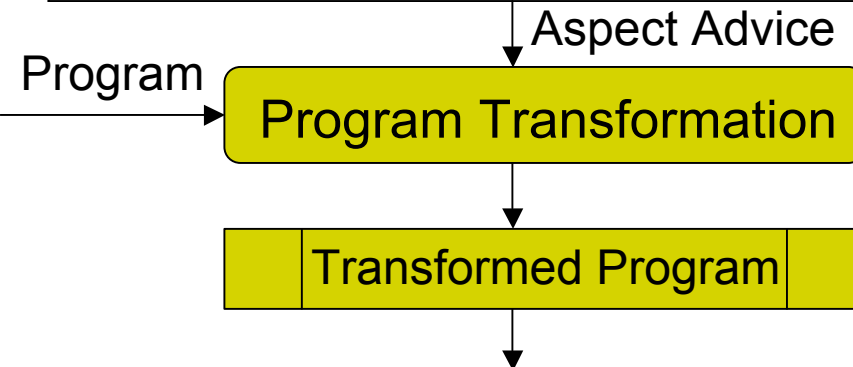
# AOP using AspectC++



# Sample AOP example with AspectC++

```
void foobar() {  
    printf("[foobar()]: Hello World!\n");  
}  
  
double foo(double z) {  
    return (1/z);  
}  
  
double bar(int x, int y) {  
    return sqrt((double)(x*x + y*y));  
}  
  
int main(int argc, char **argv) {  
    int x=4, y=5;  
    double z;  
  
    foobar();  
    z = bar(x,y);  
    printf("Value = %f\n", foo(z));  
  
    return 0;  
}
```

```
aspect example {  
    advice execution("% %(...)") : around()  
        cout << "Inside \"" << JoinPoint::signature()  
            << "\" with " << JoinPoint::ARGS << " args\n";  
        tjp -> proceed();  
    }  
};
```



```
Inside "int main(int,char * *)" with 2 args  
Inside "void foobar()" with 0 args  
[foobar()]: Hello World!  
Inside "double bar(int,int)" with 2 args  
Inside "double foo(double)" with 1 args  
Value = 0.156174
```

Output from executing transformed program

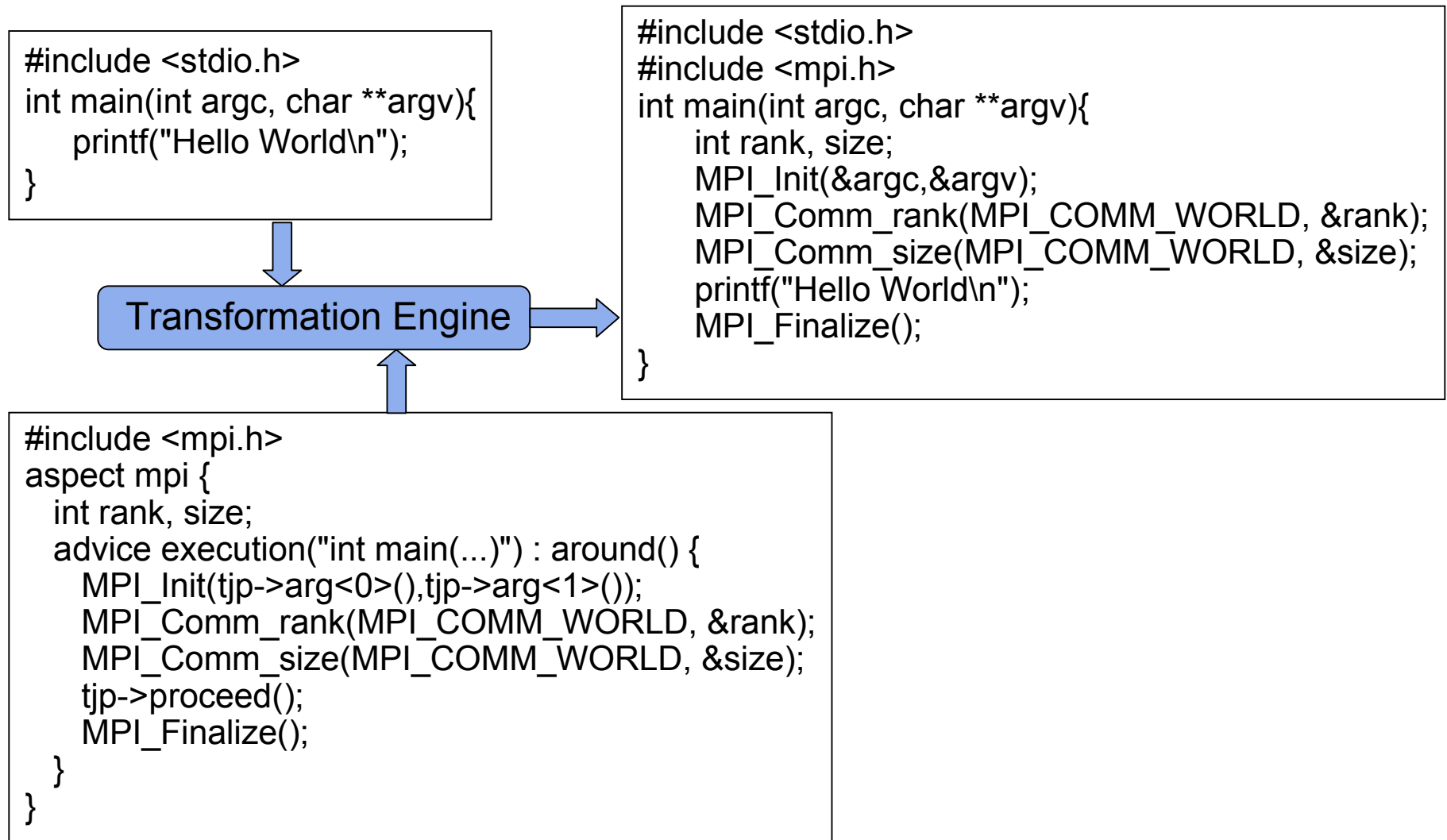
# AOP for Grid Applications

- Useful tool for separation of concerns
- Could be used for
  - Debugging
  - Profiling
  - Checkpointing
  - Inserting distributed programming logic
  - Address error handling and fault-tolerance
  - Assemble application components and middleware components
- No loss in performance or functionality, as weaving is a compile time operation

# Initial Experiments and Results

---

# Parallel Program Synthesis



# Parallel Matrix Matrix Multiplication

```
Setup()
for (k = 0; k < P; k++) {
    Preprocess()
    Multiply local blocks A and B and accumulate it in C
    Postprocess()
}
Cleanup()
```

## Cannon's Algorithm:

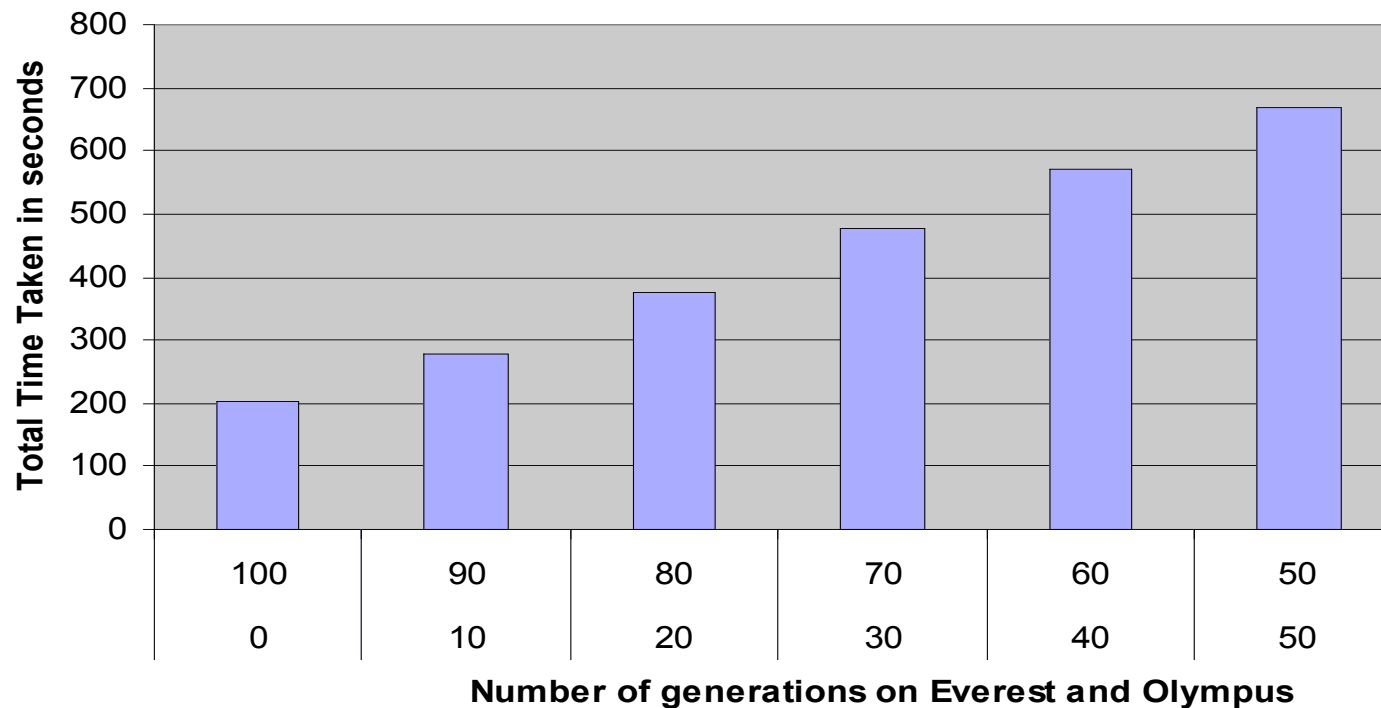
```
Setup()={Shift A p times along process row; Shift B q times along process column}
Postprocess() = {Shift A left along process row; Shift B up along process column}
Preprocess() = Cleanup() = {}
```

## Broadcast-Multiply-Roll Algorithm:

```
Preprocess()={Broadcast A along processes in the row}
Postprocess() = {Shift/Roll B along processes in the column}
Setup() = Cleanup() = {}
```

# Content Based Image Retrieval Application

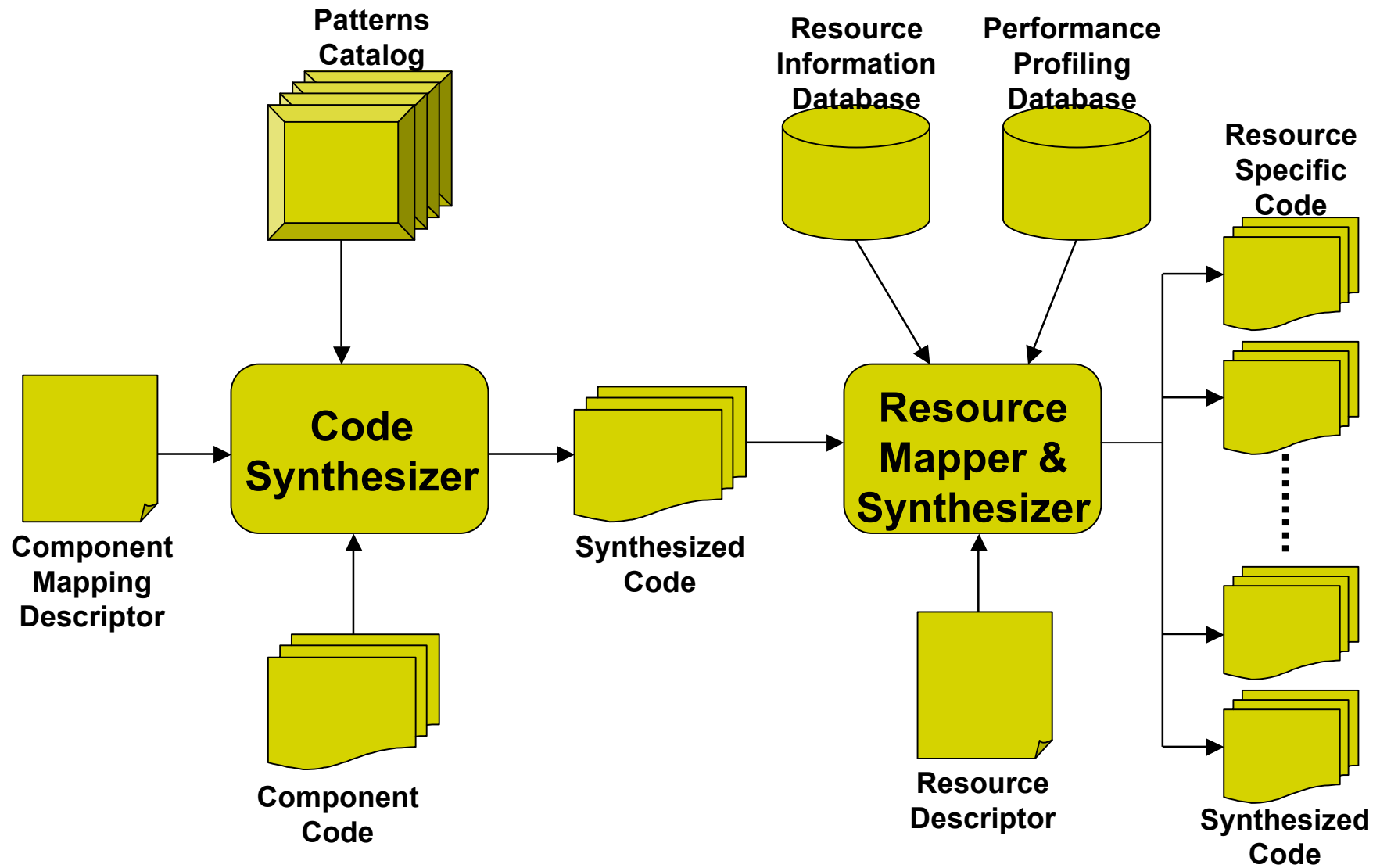
- Parallel version with checkpointing generated from sequential version using AOP
- Application migrated at different number of generations



# AOP Limitations

- AOP support focused mainly on Java (e.g., AspectJ)
- Popular C/C++ aspect weavers - AspectC and AspectC++
- Aspect weaving only supported around function calls
- Advanced program transformation tools required to support HPC applications

# Putting it all together



# Conclusions

- Classifying applications into separate categories simplifies the development and deployment process
- Developing and deploying applications on the grid presents several cross-cutting concerns
- Code generation using AOP, design patterns, and performance profiling techniques has provided encouraging results
- Integration of the individual techniques to develop a robust system currently underway

# Contact Information

---

puri@cis.uab.edu

<http://www.cis.uab.edu/ccl>