The Pilot Approach to Cluster Programming in C

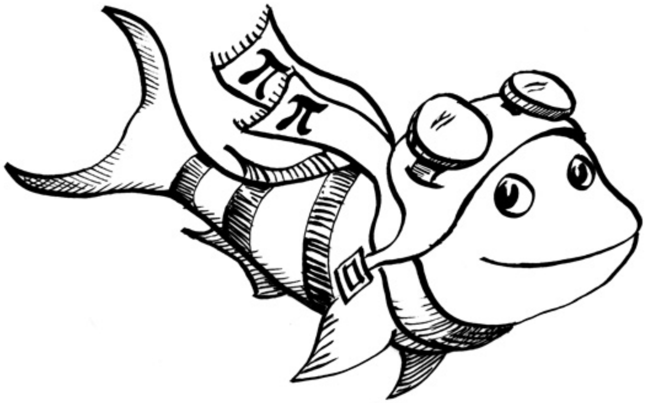
J. Carter, **Bill Gardner**, G. Grewal

Modeling & Design Automation Group

Dept. of Computing & Information Science

University of Guelph

Ontario, Canada



Outline

- Introduction & relationship to MPI
- Abstractions for parallel program design
- Programming with Pilot API
- Implementation overview
- Integrated deadlock detection
- Performance
- Experiences & future work

Introduction

- Pilot is a C library for SPMD-style, message-passing cluster programming
  - Latest version introduces Fortran API
- Target audience: novice HPC programmers, scientific programmers
  - Goal: break down barriers to adopting HPC
- Features:
  - Simple abstractions for parallel program design
  - Small, easy-to-remember API
  - Built-in deadlock detection

Relationship to MPI

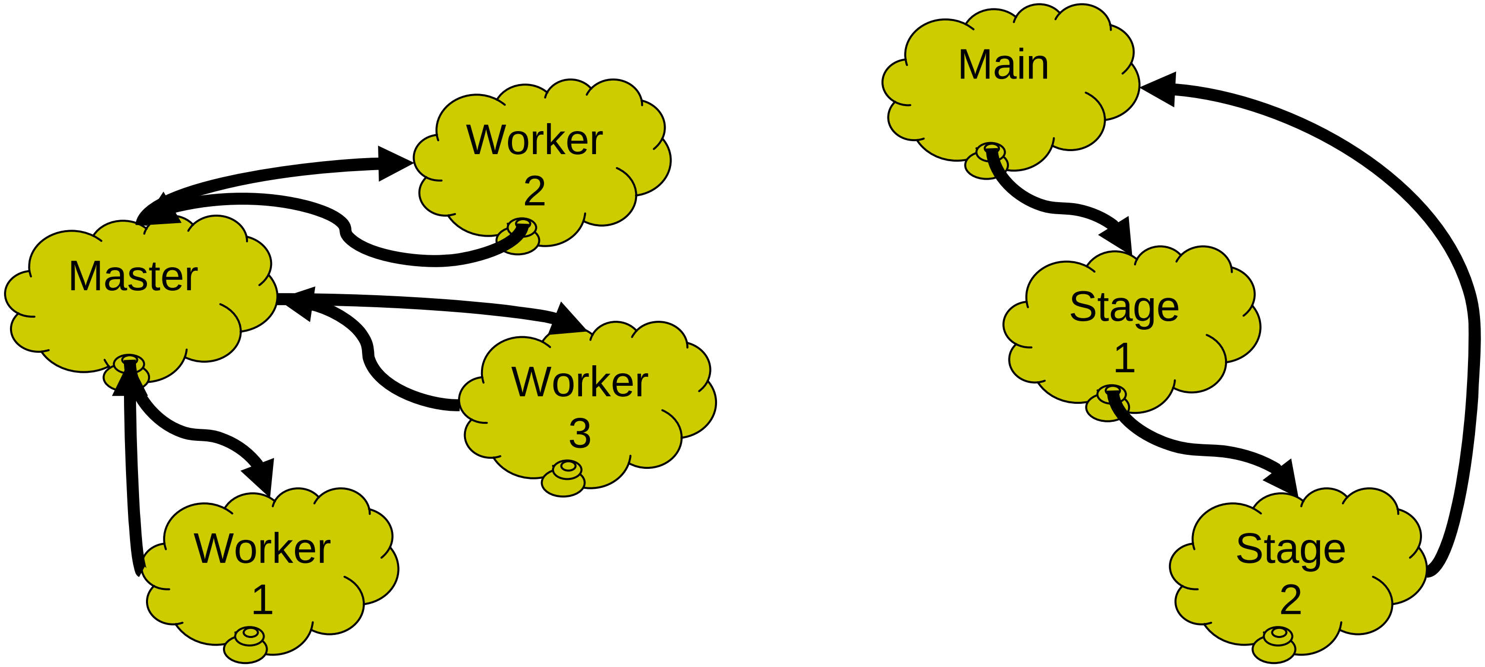
- Not intended to “replace MPI”
  - Built as thin layer on top of any standard MPI
- Purposes
  - Simpler way to teach message-passing programming
  - May be “good enough” for novice programmer
    - But still suitable for realistic applications in own right (not a toy)
    - Applications mentioned below
  - Can serve as “ramp” to transition novice to MPI if/when they require more advanced functionality

Theoretical basis

- Pilot embodies the **process** and **channel** abstractions of Communicating Sequential Processes (CSP) formal process algebra
- Users *design* solutions based on process/channel architecture before they code
- Easy to translate design into Pilot function calls
- Users need not know CSP (concepts kept under the hood)

Using process/channel design

- Visualize the organization of your algorithm
  - Draw processes to divide up your work
  - Draw channels, using arrows since they're directional



stdio metaphor

- How to make Pilot functions simple, easy to remember?
  - Engineered to conform to fprintf/fscanf syntax
  - Printf: “Most common method of debugging”
  - Even novice C programmer will be familiar
- Channel objects *like* FILE* variable/array
- Message list *like* format string

Example →

Simple code sample

#include “pilot.h”

- *Create 2 processes, blue and green:*

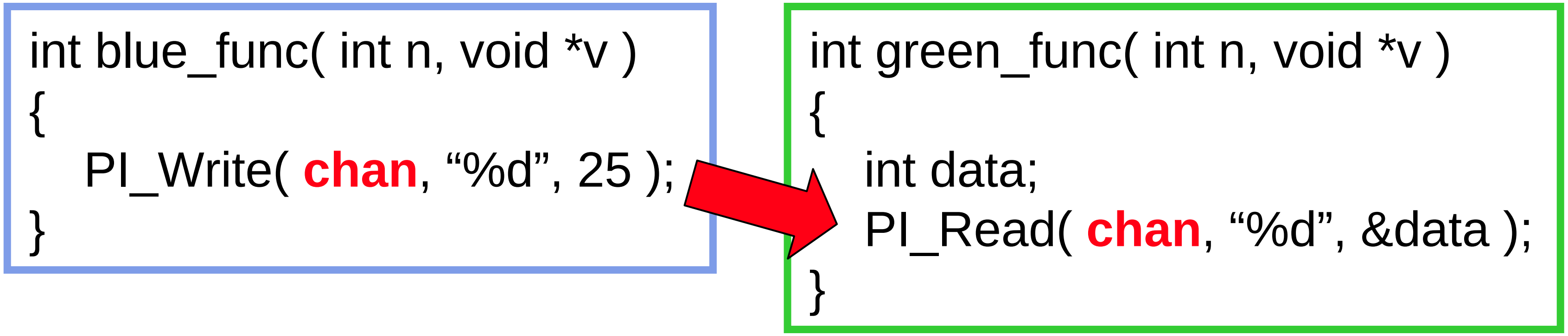
  PI_PROCESS ***blue** = PI_CreateProcess(**blue_func**, 0, NULL);

  PI_PROCESS ***green** = PI_CreateProcess(**green_func**, 0, NULL);

  - Like POSIX pthread_create(), function can execute multiple processes

- *Create a channel from blue to green:*

  PI_CHANNEL ***chan** = PI_CreateChannel(**blue**, **green**);



```
int blue_func( int n, void *v )
{
   PI_Write( chan, “%d”, 25 );
}
```

```
int green_func( int n, void *v )
{
   int data;
   PI_Read( chan, “%d”, &data );
}
```

Comparing APIs

Goal: send an array of 12 *float* coefficients, 888 *double* data values, and the 888 length

float coeffs[12]; double data[1000]; int len = 888;

- Pilot version:

```
PI_Write( chan, “%12f %*lf %d”, coeffs, len, data, len );
```

- MPI version:

```
MPI_Send( coeffs, 12, MPI_FLOAT, dest, tag, comm );
MPI_Send( data, 888, MPI_DOUBLE, dest, tag, comm );
MPI_Send( len, 1, MPI_INT, dest, tag, comm );
```

Benefits

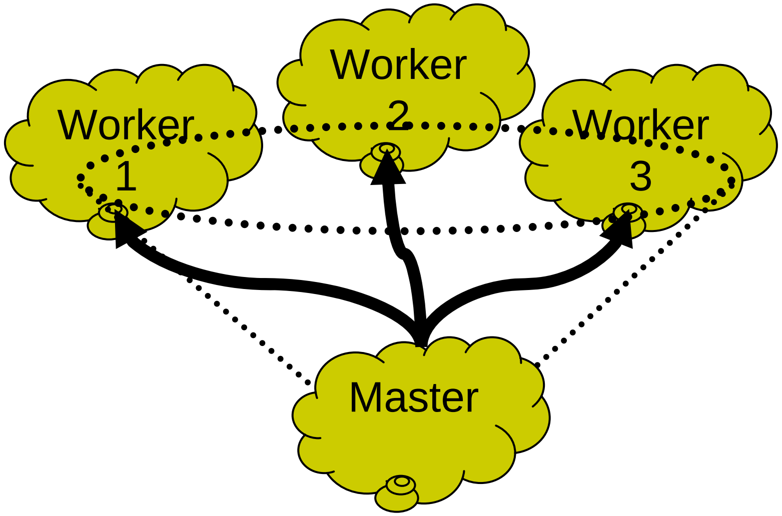
- Eliminates ability to commit some kinds of communication errors
  - No low-level arguments (dest, tag, communicator)
  - Removes some (not all) deadlock opportunities
- Messages allowed to flow between designated processes only
  - Pilot detects channel not bound to calling process, process at “read” end trying to write, etc.
- Channels not typed → data type not checked
  - Avoids undue proliferation

Collective abstraction

- MPI (and underlying hardware) *may* implement collective operations with special efficiency
  - How to take advantage in Pilot without breaking the CSP-based model?
- Solution: Add one more abstraction
  - Arbitrary *group* of channels → **bundle**
  - Must have one common process endpoint

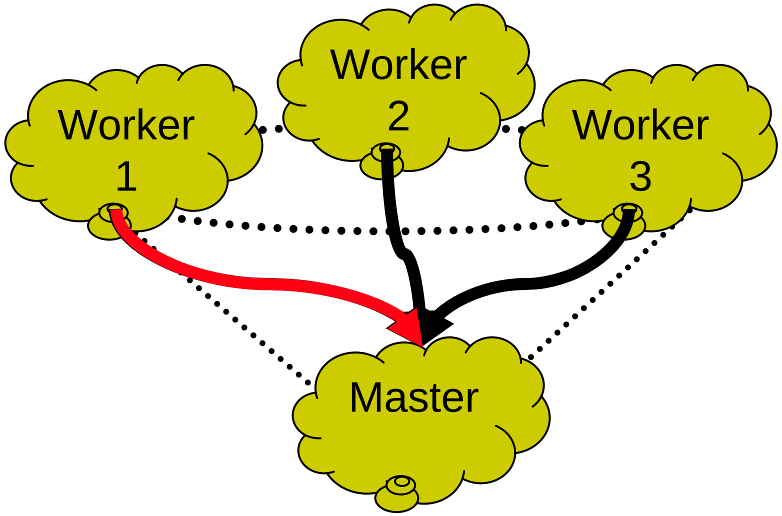
Bundle design

- “Cone” denotes the bundle of channels



Broadcasting to workers

(“Master” is a role; does not have to be rank 0 process)



Selecting on channels from workers:
Which has data to read?

Bundle functions

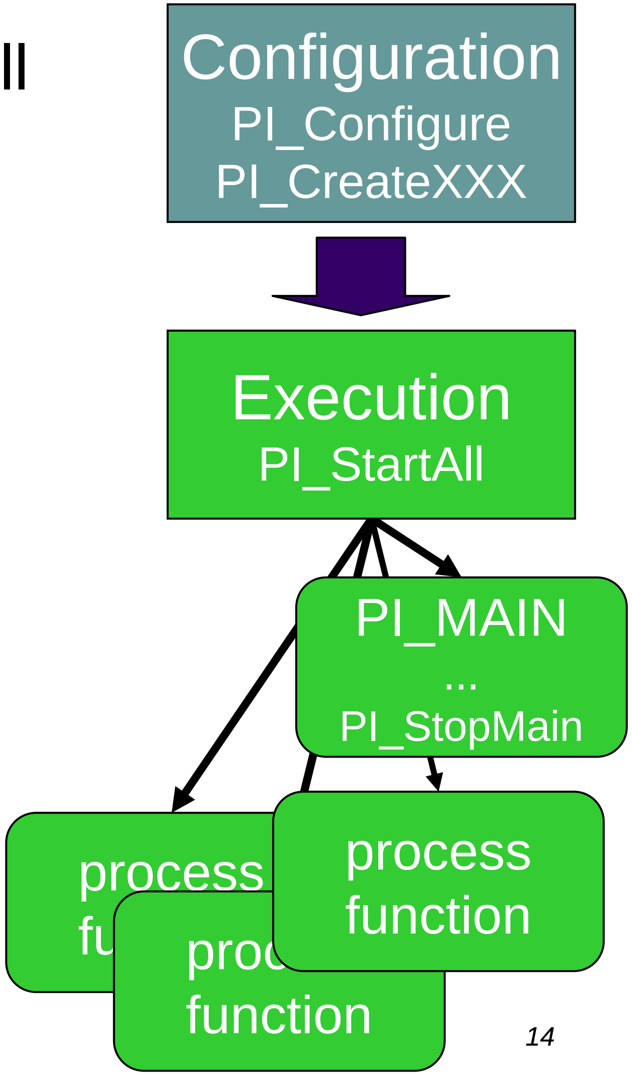
- Concept:
  - In MPI, why do we code MPI_**Bcast** in a process that is *receiving* data??
    - Rationale lies in pure SPMD approach
    - Not obvious for novice programmer to grasp
- Pilot draws a veil over this peculiarity

Master *N* Workers

  - PI_Broadcast(bundle) → PI_Read(channel$_i$)
  - PI_Gather(bundle) ← PI_Write(channel$_i$)
  - n = PI_Select(bundle) ← PI_Write(channel$_i$)
    PI_Read(channel[n])

Pilot skeleton program

- **Configuration phase** (executed on all processors):
  - Interprets command line args, starts MPI
  - PI_CreateProcess … Channel … Bundle
- **Execution phase:** PI_StartAll
  - Each Pilot process begins executing its associated function
    - Exits by returning from function
  - main() continues as rank 0 process (aka PI_MAIN)
    - Exits by calling PI_StopMain



Configuration

PI_Configure

PI_CreateXXX

Execution

PI_StartAll

PI_MAIN

…

PI_StopMain

process fu

proc function

process function

Implementation overview

- Pilot processes → *MPI processes*
- Pilot channels → *MPI tags*
- Pilot bundles → *MPI communicators*
- Extensive runtime error checking
  - Diagnostic prints out file name/line no. of Pilot call
  - Level of checking can be turned lower for less overhead

Deadlock: common parallel programming hazard

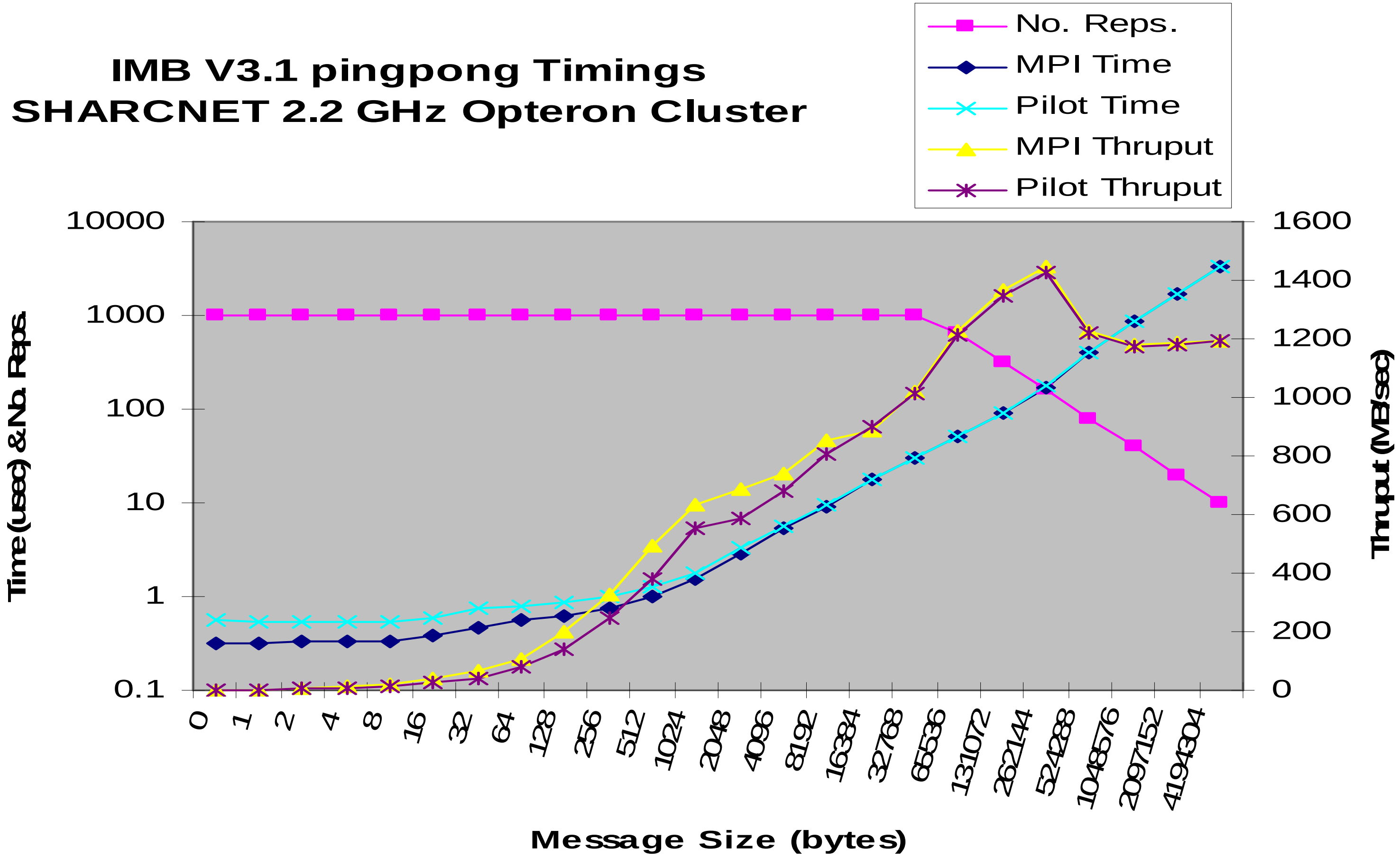
- Deadlock exacts a harsh penalty
  - MPI program typically keeps running till time budget exhausted
  - Baffling for novice users to diagnose
- Concept: *“Why do users resist tools?”*
    - DeSouza & Squyres ‘05, “Why MPI Makes You Scream!”
  - Many barriers to overcome re 3rd-party tools (e.g., deadlock detector like Umpire)
  - May not be installed on your system
  - May cost $$$
  - (Feared to) involve additional serious learning curve

Pilot’s deadlock detection

- Solution: integrate D/D into Pilot
  - Trivial to turn on: “-pisvc=d” command line arg
  - Consumes one additional MPI process
  - Aborts program with diagnosis of Pilot function calls involved in:
    - Deadly embrace
    - Circular wait
    - “Dead” wait (other end of channel process exited)
- Since Pilot functions utilize tiny subset of MPI, D/D implementation much less complex

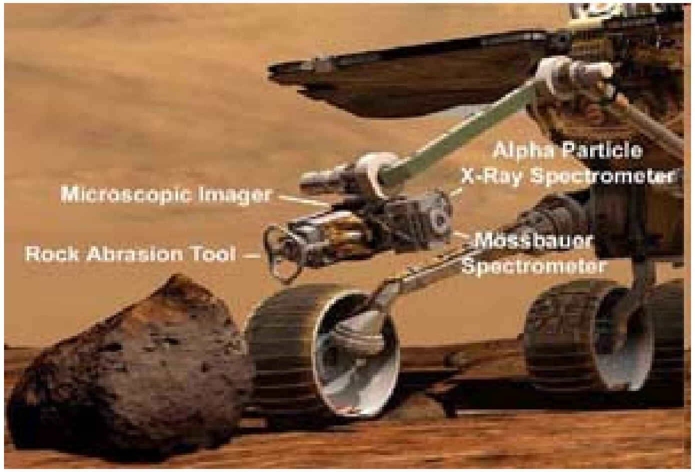
Performance

- Pilot adds very little overhead to MPI



Experiences

- Pilot development sponsored by SHARCNET consortium (SW Ontario)
- Pilot used for graduate HPC course {Guelph, McMaster, Brock, UOIT}
  - Projects: parallel MRI reconstruction, scatter search metaheuristic
- Pilot used for undergrad parallel programming course
  - Fortran: Mars Rover “search for water” spectroscopy simulation

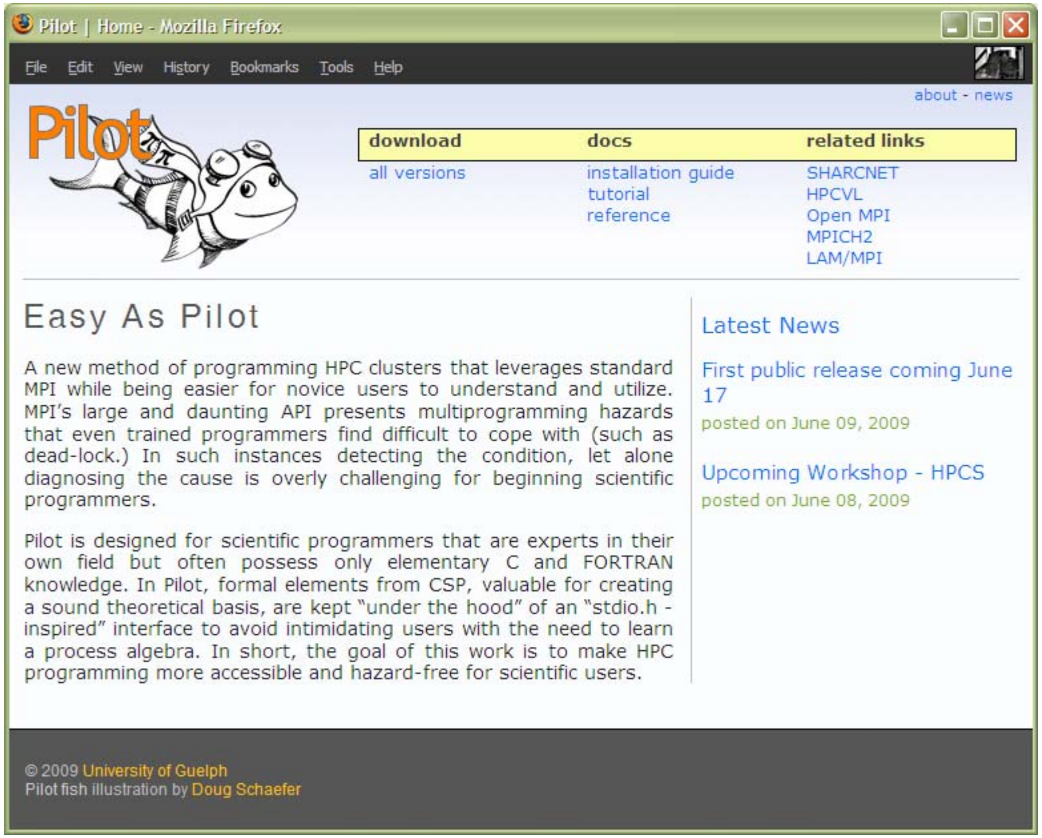


Future work

- More collective functions
- More performance measurement
- Usability study (we *think* it’s easier to use…)
- Applying to mixed cluster of Cell BE’s and other platforms
  - SHARCNET has, but almost no one using it
  - Pilot for intra-Cell PPE ←→ SPE, and inter-node communication
  - Programmer uses same process/channel paradigm rather than two or three different libraries
- Formal verification (based on CSP)

Home page: http://carmel.cis.uoguelph.ca/pilot

- Free download, install guide, tutorial
- Fortran API tested with Intel and Sun Studio
  - Uses ISO_C_BINDING
- Upcoming Toronto area tutorials
  - SHARCNET Summer School, May 31
  - HPCS, June 5


Pilot | Home - Mozilla Firefox
File Edit View History Bookmarks Tools Help
about - news
Pilot

download	docs	related links
all versions	installation guide tutorial reference	SHARCNET HPCVL Open MPI MPICH2 LAM/MPI

Easy As Pilot

A new method of programming HPC clusters that leverages standard MPI while being easier for novice users to understand and utilize. MPI's large and daunting API presents multiprogramming hazards that even trained programmers find difficult to cope with (such as dead-lock.) In such instances detecting the condition, let alone diagnosing the cause is overly challenging for beginning scientific programmers.

Pilot is designed for scientific programmers that are experts in their own field but often possess only elementary C and FORTRAN knowledge. In Pilot, formal elements from CSP, valuable for creating a sound theoretical basis, are kept "under the hood" of an "stdio.h - inspired" interface to avoid intimidating users with the need to learn a process algebra. In short, the goal of this work is to make HPC programming more accessible and hazard-free for scientific users.

Latest News

First public release coming June 17

posted on June 09, 2009

Upcoming Workshop - HPCS

posted on June 08, 2009

© 2009 University of Guelph
Pilot fish illustration by Doug Schaefer

Significance of name

1. *Surface meaning:* one who safely guides your parallel program to its destination
2. *Nod to formalism:* π -calculus
3. *Nod to* MPI
4. *Nod to SHARCNET:* pilot fish = “friend of sharks”

Reactions

- Users (undergrad and graduate students) appreciate getting a *model* to use in designing parallel programs
- Deadlock detector uncovers mysterious hangs
  - Still have to remind them to turn it on
- Quote from scientific programmer:
  - “It's less headache to organize channels and bundles than bothering about synchronization between processors. It turned out more simple and understandable for me. PILOT fits my way of thinking.”

Just teach subset of MPI?

- Certainly valid, but…
  - Pilot process/channel abstractions teach a generalized **conceptual model** for designing a parallel solution
  - MPI programmers still have to deal with low-level arguments
  - Pilot provides helpful diagnosis for usage problems, including integrated deadlock checking

IP Status

- Not (yet) open source
  - Source code copyright by Univ. of Guelph
  - Free for anyone to download/use
  - Prefer to control development at early stage to avoid…
    - bloating of API
    - breaking underlying formalism
  - Eventual open source release planned