**Message Passing Interface**
- MPI - Message Passing Interface
  - Standard for HPC programming
  - MPI-1 (1994)
    - Static processes creation
  - MPI-2 (1998)
    - Extend MPI-1 features
      - Dynamic process creation
    - Dynamic process creation (MPI_Comm_spawn())
      - Remote memory access
      - Parallel I/O
      - Recently implemented

**On-line Scheduling of MPI-2 Programs with Hierarchical Work Stealing**

Guillaume F. Rezzi
Marcie C. Cena
Elton K. Mattson
Stéphane Hudault
Philippe A. Naveau


**Outline**
- How to program with dynamic processes?
  - Recursive-like programming model
  - Divide and Conquer (D&C)
  - Recursive scheduling of dynamic programs
    - Work Stealing
      - Efficient to schedule D&C programs, used in Cilk and Satin
      - How to adapt the Work Stealing algorithm to MPI-2 context?
    - Using a Hierarchical structure
    - This work presents an on-line scheduling strategy based in Work Stealing, called Hierarchical Work Stealing
  - Divide and Conquer (D&C) Model
    - Divide Phase:
      - Problem is recursively divided into subproblems until their solutions become local

**Claim**
- It is hard to schedule MPI-2 programs efficiently
  - There are static and dynamic creation of processes and tasks
  - But you can do it with Hierarchical Work Stealing
  - This improves the performance of the application

**Message Passing Interface**
- MPI - Message Passing Interface
  - Standard for HPC programming
  - MPI-1 (1994)
    - Static processes creation
  - MPI-2 (1998)
    - Extend MPI-1 features
      - Dynamic process creation (MPI_Comm_spawn())
      - Remote memory access
      - Parallel I/O
      - Recently implemented

**Outline**
- How to program with dynamic processes?
  - Recursive-like programming model
  - Divide and Conquer (D&C)
  - Recursive scheduling of dynamic programs
    - Work Stealing
      - Efficient to schedule D&C programs, used in Cilk and Satin
      - How to adapt the Work Stealing algorithm to MPI-2 context?
    - Using a Hierarchical structure
    - This work presents an on-line scheduling strategy based in Work Stealing, called Hierarchical Work Stealing
  - Divide and Conquer (D&C) Model
    - Divide Phase:
      - Problem is recursively divided into subproblems until their solutions become local

**Message Passing Interface**
- MPI - Message Passing Interface
  - Standard for HPC programming
  - MPI-1 (1994)
    - Static processes creation
  - MPI-2 (1998)
    - Extend MPI-1 features
      - Dynamic process creation (MPI_Comm_spawn())
      - Remote memory access
      - Parallel I/O
      - Recently implemented
MPI-2 Dynamic Programs

- Divide and Conquer (D&C) Model
  - Conquer Phase:
    - The sub-problems solution are merged to compose the original problem solution.

Scheduling D&C with Work Stealing

- Work Stealing ([Blumofe and Leiserson, 1998])
  - Work Stealing works with strict computations
  - Each processor maintains a deque (double-ended queue)
  - Processor pushes a task to compute on top of the deque

Scheduling D&C with Work Stealing

- Work Stealing ([Blumofe and Leiserson, 1998])
  - Processor P_i gets idle and its deque is empty; starts stealing
  - Processor P_j chooses a task in its deque

Outline

- Motivation
- MPI-2 Dynamic Programs
- Hierarchical Work Stealing
  - Hierarchical Work Stealing Algorithm
  - Experimental Evaluation
- Conclusion
Outline of the HWS algorithm

- The CPUs will run:
  - Workers processes
    - They perform the computation of the tasks.
  - Managers processes
    - They handle the dequeues and route the stealing requests;
    - They are organized in a tree structure.
    - Internal node in the tree = managers
    - Leaves = workers.

HWS Algorithm

if LEAF then
  recv(parent, tasks);
  partial-result = Compute(tasks);
  send(parent, partial-result);
else
  if ROOT then
    deque.push(new_tasks);
  else
    recv(parent, tasks);
    deque.push(tasks);
  end if
  if # processors > then
    create-managers();
  else
    create-leaves();
  end if
  if ROOT then
    return(result);
  else
    send(parent, partial-result);
  end if
end if

HWS Algorithm

if LEAF then
  recv(parent, tasks);
  partial-result = Compute(tasks);
  send(parent, partial-result);
else
  if ROOT then
    deque.push(new_tasks);
  else
    recv(parent, tasks);
    deque.push(tasks);
  end if
  if # processors > then
    create-managers();
  else
    create-leaves();
  end if
  if ROOT then
    return(result);
  else
    send(parent, partial-result);
  end if
end if
HWS Algorithm

if is processor idle
    open channel to
    task = dequeue(task);
    send(task);
    wait for
    while there are children
        child = dequeue(child);
        send(task);
        if child
            break
        else
            if is processor idle
                continue
            else
                child = dequeue(child);
                send(task);
        end if
    end while
end if

else
    if is processor idle
        receive(task);
        send(task);
    else
        receive(task);
        enqueue(task);
    end if
end if

if # processors > 1
    create managers;
else
    create leaves;
end if

if is processor idle
    return(result);
else
    send(parent, partial-result);
end if

Outline

- Motivation
- MPI-2 Dynamic Programs
- Hierarchical Work Stealing
- Experimental Evaluation
  - The N-Queens Application
  - Performance: MPI vs. Satin
  - Scheduling in Dynamic Environment
  - Hierarchical Performance
- Conclusion

N-Queens Application

- Placing N queens in a NxN chessboard
  - No queen may capture any other
  - Backtracking algorithm
    - Placing recursively and exhaustively the queens, row by row

Performance of MPI with HWS vs. Satin

- Run-time of N Queens with MPI-2 vs. Satin: sequential case
  - MPI is better than Satin
  - Maintains the same scalability

- Run-time gain of the N Queens with MPI-2 over Satin: the parallel case (50-100)
  - MPI is better than Satin
  - Maintains the same scalability
Scheduling in Dynamic Environment

Number of processes on processor vs. time. A new node is added each minute.

In a distributed platform:
- MPI-2 with HWS uses dynamically the CPUs

Impact of the tree structure

For a fixed number of processors (40), different numbers of managers can be chosen.

Outline

- Motivation
- MPI-2 Dynamic Programs
- Hierarchical Work Stealing
- Experimental Evaluation
- Conclusion

Conclusion

- MPI-2 can be used to implement D&C programs
- Dynamic process creation to implement recursive calls – the divide phase
- Problem: How to schedule tasks in a dynamic and distributed way?
  - We propose Hierarchical Work Stealing
    - Work Stealing offers distributed efficient scheduling devised by Cilk
    - Hierarchical to follow the parent/children communication relationship
- HWS evaluation:
  - Better performance than Satin
  - Can be used in dynamic environments

On-line Scheduling of MPI-2 Programs with Hierarchical Work Stealing

Guilherme P. Pazzi
Manuela C. Cere
Etch N. Mathis
Victor Reisland
Philippe G. A. Navarro