Parallel Computing I

Parallel Program Designs
SMP: Massively Parallel Problems

A First Parallel Program

Primality tests via trial division

- `PrimeTesterSeq.java`
- `PrimeTesterSmp.java`

Parallel Programming

How do we *write* a program to make use of a parallel computer?

Use a standard programming language and generic OS kernel functions:

- SMP: C with `pthreads`, Java with `Thread` objects
- Cluster: C with sockets API, Java with socket classes

- Requires expertise in threads and/or sockets
- Much repetition accross parallel programs

Use a parallel programming library:

- OpenMP (SMP)
- MPI (Cluster)
- Parallel Java (SMP and Cluster)

Parallel Programming

How do we *design* a program to make use of a parallel computer?

Three patterns for designing parallel programs

- Result parallelism
- Agenda parallelism
- Specialist parallelism

Two additional patterns for implementing parallel programs

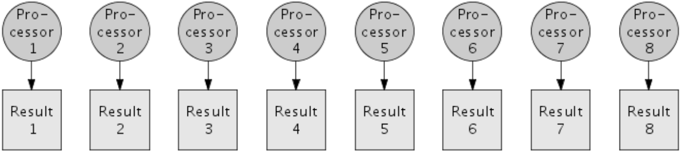
- Clumping (or slicing)
- Master-Worker

Steps for parallel program design

- Identify the pattern that best matches the problem.
- Take the pattern's suggested design as the starting point.
- Implement the design using the appropriate constructs.

Result Parallelism

- The result of the computation (a data structure) is divided into elements
- The elements are all computed simultaneously

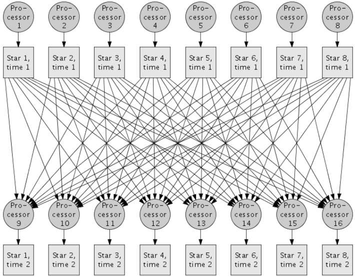


Example: calculating pixels in the frames of a computer-animated film.

Important characteristic: no dependencies between individual results.

Result Parallelism

- The result of the computation (a data structure) is divided into elements
- The elements are all computed simultaneously

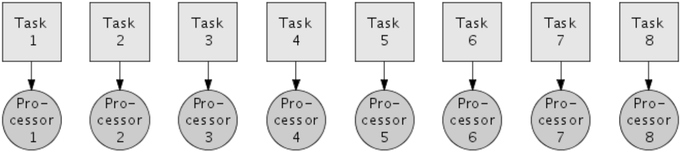


Example: calculating positions of stars in a cluster over time.

Important characteristic: some dependencies between individual results.

Agenda Parallelism

- The problem is divided into tasks (an "agenda")
- Multiple workers perform the tasks simultaneously
- Each worker is a "generalist" that can perform any task

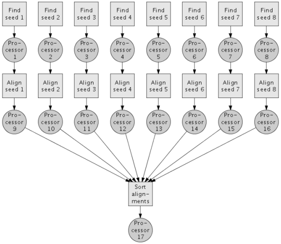


Example: querying a DNA sequence database.

Important characteristic: not really interested in individual results.

Agenda Parallelism with Reduction

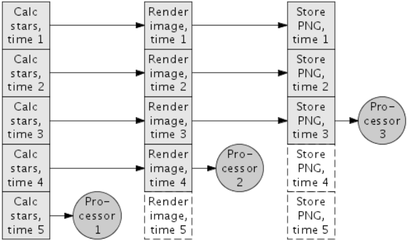
- The problem is divided into tasks (an "agenda")
- Multiple workers perform the tasks simultaneously
- Each worker is a "generalist" that can perform any task
- Output a single summary result (reduction) of individual tasks



Example: Basic Local Alignment Search Tool (BLAST)

Specialist parallelism

- The program is divided into sub-programs
- Each sub-program is a "specialist" that does a different thing
- All specialists run simultaneously



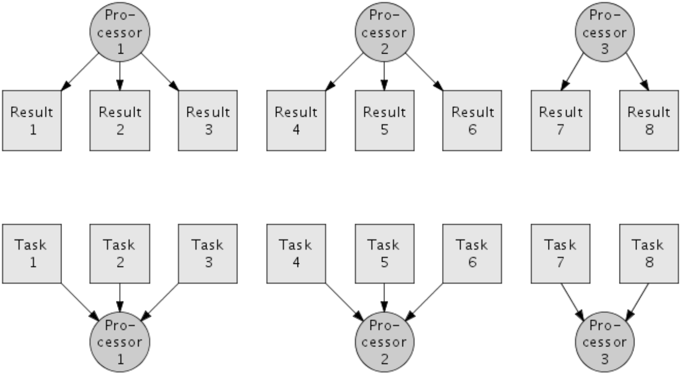
Example: animate positions of stars in a cluster over time

Important characteristic: sequential dependencies leads to pipelining
Important characteristic: independence leads to overlapping

Clumping (or Slicing)

Conceptually, every result/task assigned to a different processor.

In practice, a single processor works on multiple results/tasks.



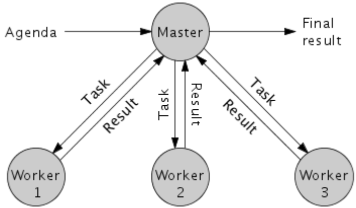
Fits well with Result and Agenda parallelism.

Master-Worker

Conceptually, every result/task assigned to a different processor.

In practice, a single processor works on multiple results/tasks.

- Master: Send task to a worker; Receive task result from any worker; Record task result; Loop.
- Worker: Receive task from the master; Compute task results; Send results to the master; Loop.



Fits well with Agenda parallelism.

Other Patterns

Are there other patterns of parallel programming?

Other Patterns

Are there other patterns of parallel programming?

Certainly!

One missing aspect of the above patterns: dynamic creation of tasks.

Another missing aspect: nested parallel tasks.

Yet another missing aspect: problems combine multiple patterns.

Other Patterns

Are there other patterns of parallel programming?

Certainly!

One missing aspect of the above patterns: dynamic creation of tasks.

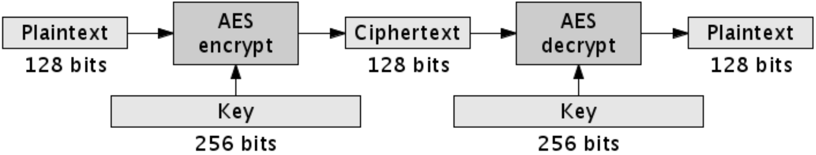
Another missing aspect: nested parallel tasks.

Yet another missing aspect: problems combine multiple patterns.

Arguably: Agenda parallelism with Master-Worker pattern is most general.

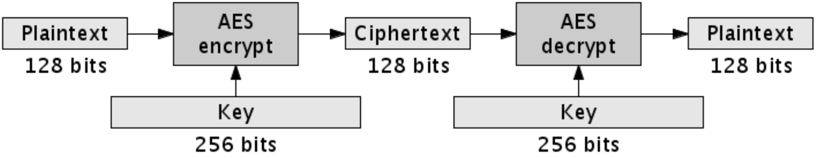
AES Encryption

Consider AES encryption:



AES Encryption

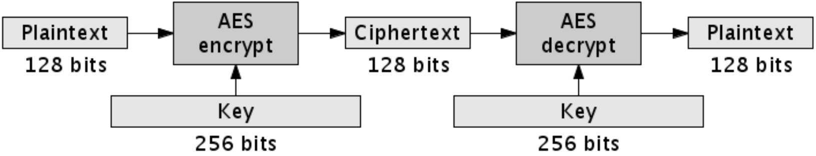
Consider AES encryption:



How can we "crack" AES encryption?

AES Encryption

Consider AES encryption:



How can we "crack" AES encryption?

Attack the encryption with a known plaintext (using exhaustive search)!

- We (somehow) obtain a plaintext block and the ciphertext block
- We (exhaustively) try `key` equal to $0, 1, \ldots, 2^{256} - 1$:
  - With each trial key, encrypt the plaintext and compare to the ciphertext.
  - (Alt: decrypt the ciphertext and compare to the plaintext.)
    - (Might not even need plaintext, if we know it when we see it.)

AES Key Search

How much computation does this plaintext attack take?

AES Key Search

How much computation does this plaintext attack take?

- On the order of $\mathbf{2^{256}}$ (or $\mathbf{10^{77}}$) encryptions.
- Universe will end before finding key,
  even if we use all computers on Earth in parallel.
- (Good for encryption!)

But what if we knew **232** bits of the **256**-bit key?

- On the order of $\mathbf{2^{24}}$ (or $\mathbf{1.67 \times 10^{7}}$) encryptions.
- Feasible to crack in minutes on one computer
- Feasible to crack in seconds on parallel computer

AES Partial Key Search:

- Inputs: a plaintext block `p`, a ciphertext block `c`, a partial key `k'` with $\mathbf{256 - n}$ bits of the `k` (which produced `c` from `p`)
- Outputs: the complete key `k` (which produced `c` from `p`)
- Algorithm: exhaustive search

AES Partial Key Search

Preparing the input:

Generate a random 256-bit key:

```
[mtf@fenrir code]$ java edu.rit.smp.keysearch.MakeKey
3f5f5a45d429feff41bf957ed46d69f7df8323926ad32fe9eb0dd07bb7cfe5b6
```

Encrypt a message and mask key:

```
[mtf@fenrir code]$ java edu.rit.smp.keysearch.Encrypt 'Hello, PC1!'\
3f5f5a45d429feff41bf957ed46d69f7df8323926ad32fe9eb0dd07bb7cfe5b6 24
48656c6c6f2c20504331210000000000
ea482cc8278b57cba021228877e6627b
3f5f5a45d429feff41bf957ed46d69f7df8323926ad32fe9eb0dd07bb7000000
24
```

See textbook for more details.

Sequential AES Partial Key Search

`FindKeySeq.java`

Parallel AES Partial Key Search

How do we convert the sequential key search program to a parallel key search program?

- What pattern does the AES partial key search use?
- Are there any sequential dependencies?
- If we have 2^n processors, then how should we divide the problem and how long should it take?

Parallel AES Partial Key Search

How do we convert the sequential key search program
to a parallel key search program?

- What pattern does the AES partial key search use?
- Are there any sequential dependencies?
- If we have 2^n processors, then how should we divide the problem and how long should it take?

This type of problem is called a *massively parallel problem* or *embarrassingly parallel problem*!

Massively Parallel Problems

Some problems require many, many independent computations.
These independent computations can be done in parallel.

For the Parallel AES Partial Key Search,
if we use K threads on K processors,
then how do we partition the problem?

Massively Parallel Problems

Some problems require many, many independent computations.
These independent computations can be done in parallel.

For the Parallel AES Partial Key Search,
if we use K threads on K processors,
then how do we partition the problem?

Each partition handles $2^n/K$ computations.

For example, if $n = 24$ and $K = 8$,
then each thread/processor handles approx. 2097152 computations.

Next task is to convert the sequential program
to a parallal program using Parallel Java.

SMP Parallel Programming with PJ

- Parallel Team
- Parallel Region
- Parallel For Loop
- Variables

Parallel Team

An SMP parallel program (in PJ) has the following structure:

- Initial setup
- Create a parallel team
- Have the parallel team execute the computation
- Clean up

Parallel Team

An SMP parallel program (in PJ) has the following structure:

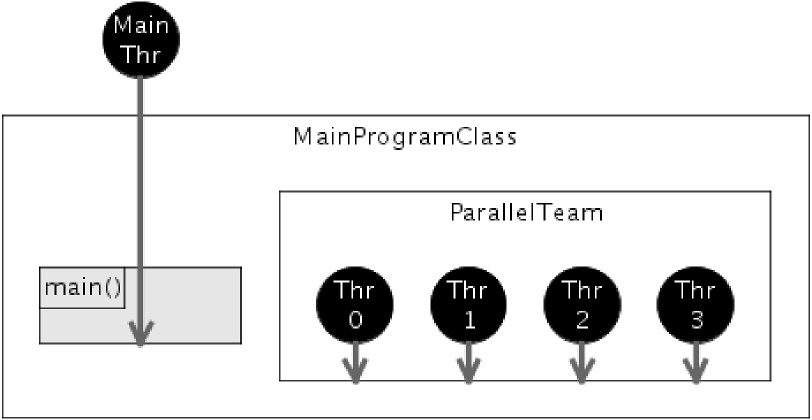
- Initial setup
- Create a parallel team
- Have the parallel team execute the computation
- Clean up

Creating parallel teams:

- `new ParallelTeam(4);`
  - create a team of four threads
- `new ParallelTeam();`
  - create a team of #processors threads (recommended)
- `$ java -Dpj.nt=8`
  - specify the number of threads

Parallel Team

Creating a parallel team introduces K new threads, in addition to the main thread.



Parallel Region

A parallel team represents the threads
that will carry out the program's computation.

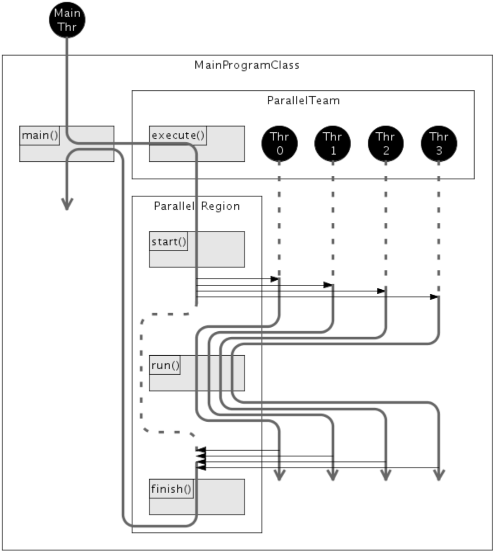
A parallel region (passed to the parallel team's `execute` method)
represents the code for the program's computation.

```
new ParallelTeam().execute(new ParallelRegion() {
  public void start() {
    // Initialization code
  }
  public void run() {
    // Parallel computation code
  }
  public void finish() {
    // Finalization code
  }
}
```

Note: convenient with an anonymous inner class.

Parallel Region

The methods of `ParallelRegion` are executed by different threads.



Parallel For Loop

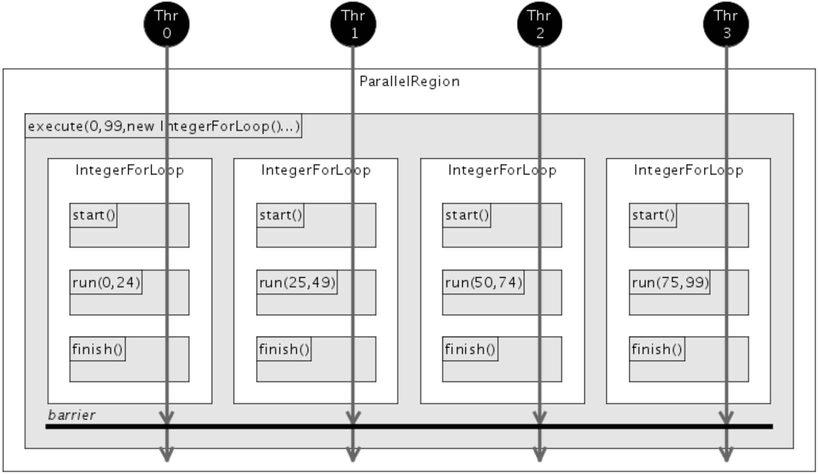
The program's computation is often a loop of (independent) iterations.

```
new ParallelTeam().execute(new ParallelRegion() {
  public void run() {
    execute(0,1023,new IntegerForLoop() {
      public void start() {
        // Per-thread per-loop initialization code
      }
      public void run(int first, int last) {
        // Loop computation code
      }
      public void finish() {
        // Per-thread per-loop finalization code
      }
    });
  }
}
```

Note: convenient with an anonymous inner class.

Parallel For Loop

The methods of `ParallelRegion` are executed by different threads.



Parallel For Loop

In more detail, execution proceeds as follows:

1. Each thread creates an instance of the `IntegerForLoop` subclass
2. Each thread calls the `execute()` method with the lower and upper index bounds.
3. The `execute()` method partitions the index range into K chunks (where K is the number of threads in the parallel team)
4. Each thread calls the `start()`, `run()`, and `finish()` methods, where the `run()` method receives a different index range

On a parallel computer with K processors, this program runs faster, because each thread executes only $1/K$ of the loop iterations.

Variables in a Parallel Java Program

A key decision in writing a parallel program is deciding where to declare variables.

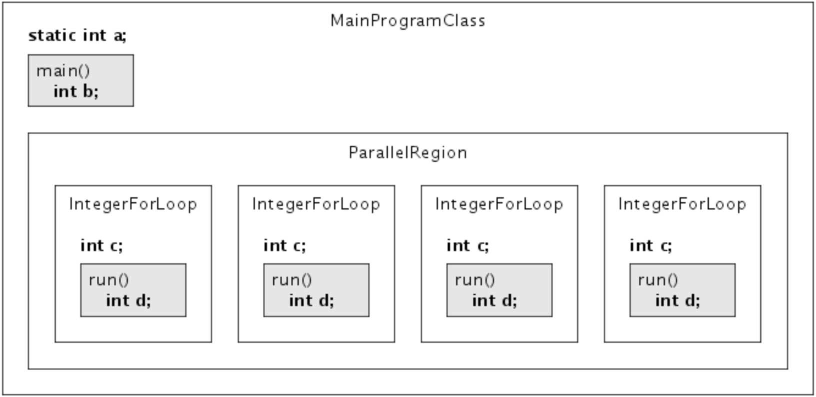
- Shared (global) variables
- Per-thread variables
- Loop local variables
- Main program local variables

Which code can access which variables?

Variables in a Parallel Java Program

```
public class MainProgramClass {
  // Shared (global) varriable declarations
  static int a;

  public static void main(String[] args) {
    // Main program local variable declarations (incl. args)
    int b;

    new ParallelTeam().execute(new ParallelRegion() {
      public void run() {
        execute(0,1023,new IntegerForLoop() {
          // Per-thread variable declarations
          int c;

          public void run(int first, int last) {
            // Loop local variable declarations (incl. i)
            int d;

            for (int i = first; i <= last; i++) {
              // Code for loop iteration i
            }
          }
        })
      }
    })
  }
}
```

Parallel Variables



Shared Memory

An SMP parallel program is organized around data structures located in shared memory.

- In a result parallel program, the shared data structure may contain all the results.
- In an agenda parallel program, the shared data structure may contain the agenda items and their results.
- In a specialist parallel program, the shared data structure may contain outputs and inputs.

In Java, shared (global) variables are located in shared memory; hence, use shared (global) variables to reference the shared data structures.

Shared Memory

Pro: a shared variable can be accessed by all threads.
Con: a shared variable can be accessed by all threads.

When multiple threads access the same shared variable, conflicts may arise:

- read-write conflict: one thread reads a variable at the same time as another thread writes the variable.
- write-write conflict: one thread writes a variable at the same time as another thread writes the variable.

Synchronize threads to avoid conflicts.

- Force a specific ordering of thread accesses

Synchronization adds overhead, so synchronize only when necessary.

SMP Parallel AES Partial Key Search

`FindKeySmp.java`