Mandelbrot

Lorenzo Schwertner Kaufmann

Initial changes

Cish

```
printf("Hello\n");

fprintf(stderr, "Error\n");

auto* pic = new uchar[size];

char* str;
```

C++ish

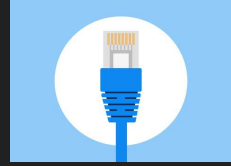
```
std::cout << "Hello" << std::endl;

std::cerr << "Error" << std::endl;

std::vector<uchar> vec(size);

std::string str;
```

Dependency tip

```
for (int frame = 0; frame < frames; frame++)
```

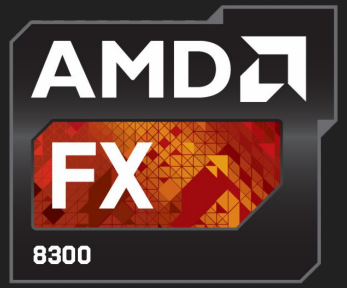
```
for (int frame = frames - 1; frame >= 0; frame--)
```

Hardware



1GBps

FX-8300

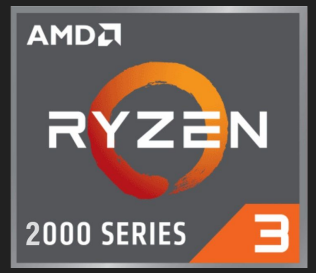


8 cores
3.3/4.2 GHz
PassMark: 5,263

i3-9100F

4 cores
3.6/4.2 GHz
PassMark: 6,778

Ryzen 3 2200G

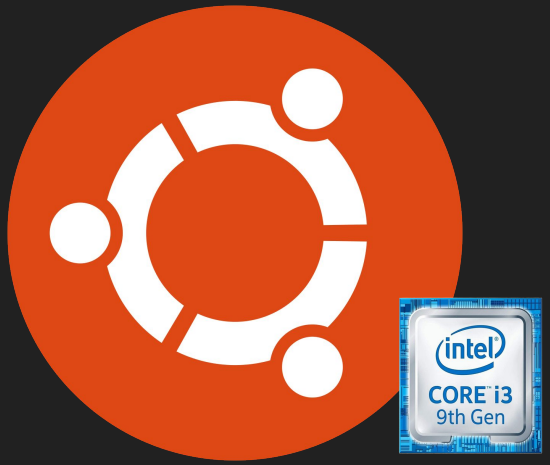


4 cores
3.5/3.7 GHz
PassMark: 6,766

OS

WSL2/Debian

Ubuntu

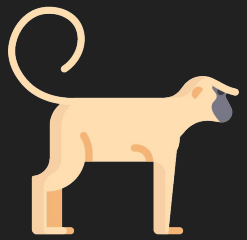


WSL2/Debian

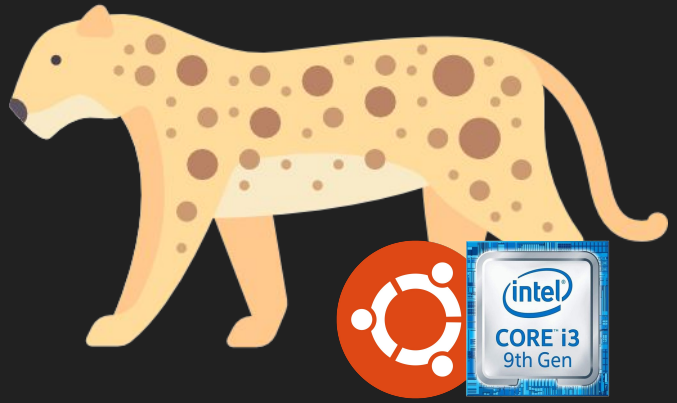
MPICH 3.4

Names

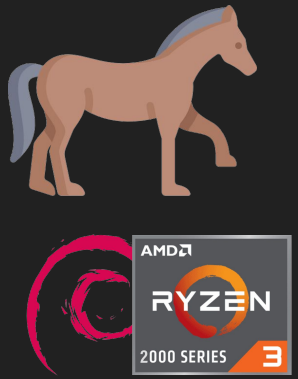
Monkey



Leopard



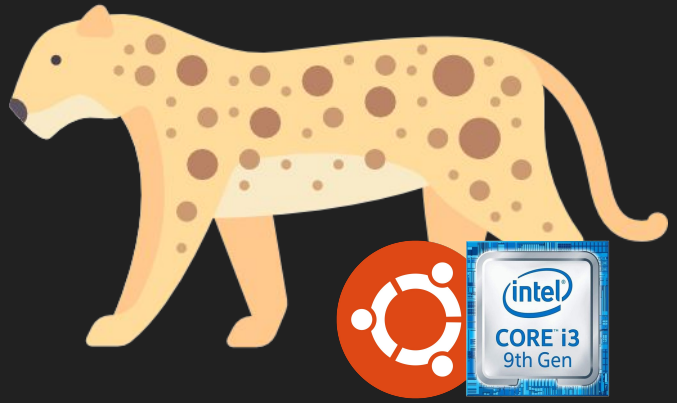
Horse



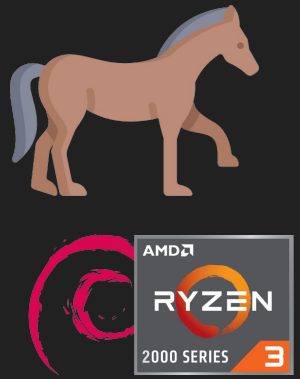
Names

Monkey

Leopard



Horse



Architecture

```
if (myRank == 0)
{
   MandelbrotManager::Start(argc, argv);
}
else
{
   MandelbrotWorker::Start();
}
```

Manager / Dispatcher

```
for (int i = 0; i < workers; i++)
{
   int rank = i + 1; // Rank zero is reserved for manager

   int start = i * frames / workers;
   int end = i == workers - 1 ? frames - 1 : (i + 1) * frames / workers - 1;

   int data[3] = {start, end - start, resolution};
   MPI_Send(data, 3, MPI_INT, rank, PacketTag::Request, MPI_COMM_WORLD);
}

[…]
```

Manager / Receiver

```
[…]

int frameIndex = 0;
for (int i = 0; i < workers; i++)
{
   int rank = i + 1; // Rank zero is reserved for manager

   int start = i * frames / workers;
   int end = i == workers - 1 ? frames - 1 : (i + 1) * frames / workers - 1;

   int workerFrames = end - start;
   int frameSize = resolution * resolution;
   int bufferSize = workerFrames * frameSize;
   std::vector<uchar> batch(bufferSize);
   MPI_Recv(batch.data(), bufferSize, MPI_UNSIGNED_CHAR, rank, PacketTag::Reply,
       MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   // Write BMP's
}
```

Worker

```
int data[3];
MPI_Recv(&data, 3, MPI_INT, 0, PacketTag::Request, MPI_COMM_WORLD,
     MPI_STATUS_IGNORE);
int start = data[0];
int frames = data[1];
int resolution = data[2];
std::vector<uchar> batch(frames * resolution * resolution);
float delta = 0.001f;
float deltaAcceleration = 0.98f;
for (int i = 0; i < start; i++)
{
   delta *= deltaAcceleration;
}
MandelbrotGenerator::Generate(batch, delta, deltaAcceleration, frames, resolution);
MPI_Send(batch.data(), (int) batch.size(), MPI_UNSIGNED_CHAR, 0, PacketTag::Reply,
     MPI_COMM_WORLD);
```

Results

Method/Problem	256x256x24	1024x1024x120
Original	1422 ms	117 s
Leopard (2)	700 ms	62.3 s
Leopard (4)	543 ms	43.3 s
Leopard(4) + Horse(4)	**426 ms**	**27.5 s**

Comparison between MPI and original

Results (Speedup)

Method/Problem	256x256x24	1024x1024x120
Original	1422 ms	117 s
Leopard (2)	700 ms (2.03x)	62.3 s (1.87x)
Leopard (4)	543 ms (2.61x)	43.3 s (2.70x)
Leopard(4) + Horse(4)	**426 ms** (3.33x)	**27.5 s** (4.25x)

Comparison between MPI and original

Results (Efficiency)

Method/Problem	256x256x24	1024x1024x120
Leopard (2)	101%	93%
Leopard (4)	65%	67.5%
Leopard(4) + Horse(4)	41%	53%

Comparison between MPI and original

Results

- Not so good!
- Not optimized (DEBUG compilation)
- Probably due to:
  - Sequential Receive
  - Horse < Leopard (50%)
  - Considerable Data Throughput (125 MB)
  - MPICH Lack of Tweaking!?
- Short time
- Hard configuration

My Mandelbrot