Optimising Genomics Codes on the Cray MTA-2

Jon Gibson

jon.gibson@man.ac.uk

The Project

The aim of the project is to optimise the genomics codes Phrap and Cross-Match for performance on the Cray MTA-2.

Currently involved with the project are:

- Jon Gibson @ CSAR,
- Keith Taylor @ CSAR and
- Jim Maltby @ CRAY.

The Cray MTA-2

- **M**ulti-**T**hreaded Multiple active threads (up to 128) on each processor.
  - Used to hide latency.
  - 16 to 256 processors.
- Scalable uniform access to global shared memory.
  - 2.4GB/s bandwidth.
  - 4GB of memory per processor.
- Easy programming model.
  - Mainly loop-based parallelism.
  - Uniform access to global memory.
  - Dynamic scheduling of tasks.

Phrap and Cross-Match

- **Phr**agment **a**ssembly **p**rogram
- Assembles shotgun DNA sequence data.
  - Typically, single input file containing the reads.

- **Cross-Match**
- Compares any two sets of (long or short) DNA sequences.
  - Typically, 2 input files: the *query* sequences and the *subject* sequences.

- Both programs use a "banded" version of SWAT, an efficient implementation of the **S**mith-**Wat**erman algorithm.

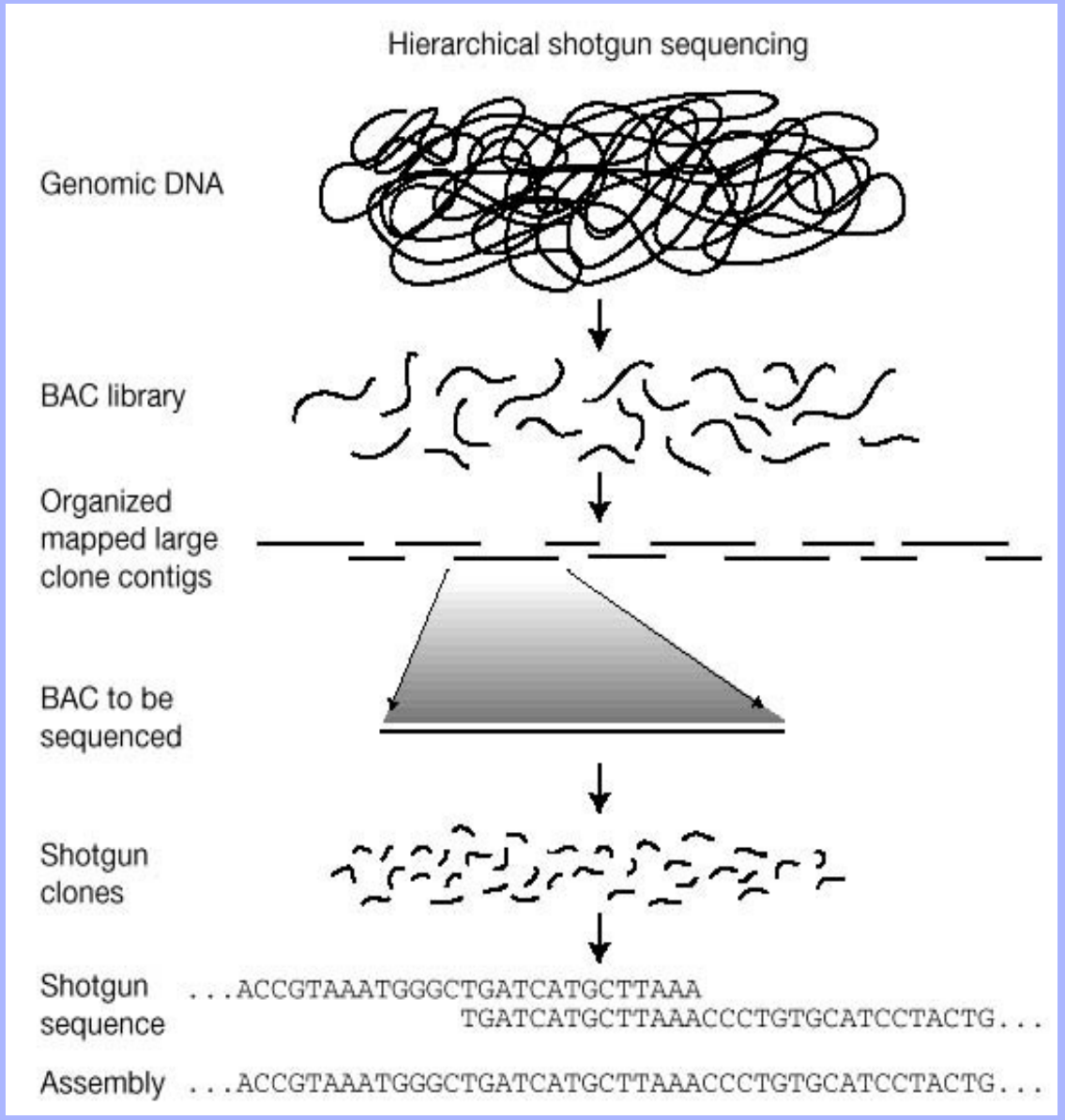
Smith-Waterman Algorithm

- Algorithm for finding the optimal LOCAL alignment of any two sequences
- Iterative matrix-based calculation
  - All possible pairs of residues, one from each sequence, are represented in a 2D array of cells.
  - All possible alignments are represented by pathways through this array.

Smith-Waterman Algorithm

- The highest score is assigned to each cell out of those of all the possible paths/alignments leading to it.
  - Based on the previous score, the similarity score of the pair of residues and gap penalties.
- The alignment is determined by tracing the pathway back from the highest scoring cell.

Hierarchical Shotgun Sequencing



Cross-Match

- Read in sequence and quality data.
- Find pairs of reads having matching words of a given minimum length. Each such word defines a band in the Smith-Waterman matrix, centred on the word match.
- Eliminate exact duplicate reads. Do swat comparisons of pairs of reads which have matching words and compute the (complexity-adjusted) swat score.
- Print the matches.

Previous Work on the Code

- Modifications had already been made to the code so that it would run in parallel on a number of different architectures. Code applicable to a given architecture was picked out using *#ifdef* statements. An example, for an SGI machine, is considered on the next slide. The slide after this shows how this would be coded on the MTA.

Sample Code on the SGI

```
#ifdef SGI
#define     thread_num()  mp_my_threadnum()
#endif
……
#ifdef SGI
#pragma parallel
#pragma local (entry1,tempseq,i)
 {
#pragma pfor iterate (entry1=ies; ief-ies+1; 1)
#pragma schedtype (dynamic)
#endif
 for (entry1 = ies; entry1 <= ief; entry1++) {
#ifdef SGI
      i = thread_num();
#endif
……{body of loop}……
```

Code Ported to the MTA

```
#ifdef MTA
  __sync int lock$[MAX_PROCS];  int temp;
#endif
…….
#ifdef MTA
#pragma mta assert parallel
#endif
    for (entry1 = ies; entry1 <= ief; entry1++) {
#ifdef MTA
#pragma mta assert local tempseq, i, temp
      i = entry1%nprocs;
      lock$[i] = 1;
…….{body of loop}……..
#ifdef MTA
      temp = lock$[i];
#endif
```

Canal Compiler Analysis of this MTA Code

```
|       |#ifdef MTA
 |       |#pragma mta assert parallel
 |       |#endif
 |       |    for (entry1 = ies; entry1 <= ief; entry1++) {
 |       |#ifdef MTA
 |       |#pragma mta assert local tempseq, i, temp
 | 12 p |       i = entry1%nprocs;
 | 12 D |       lock$[i] = 1;
 |       |#endif
 | 12 p |       tempseq = get_seq(entry1);
 | 12 D |       find_scores (entry1, tempseq, i);
```

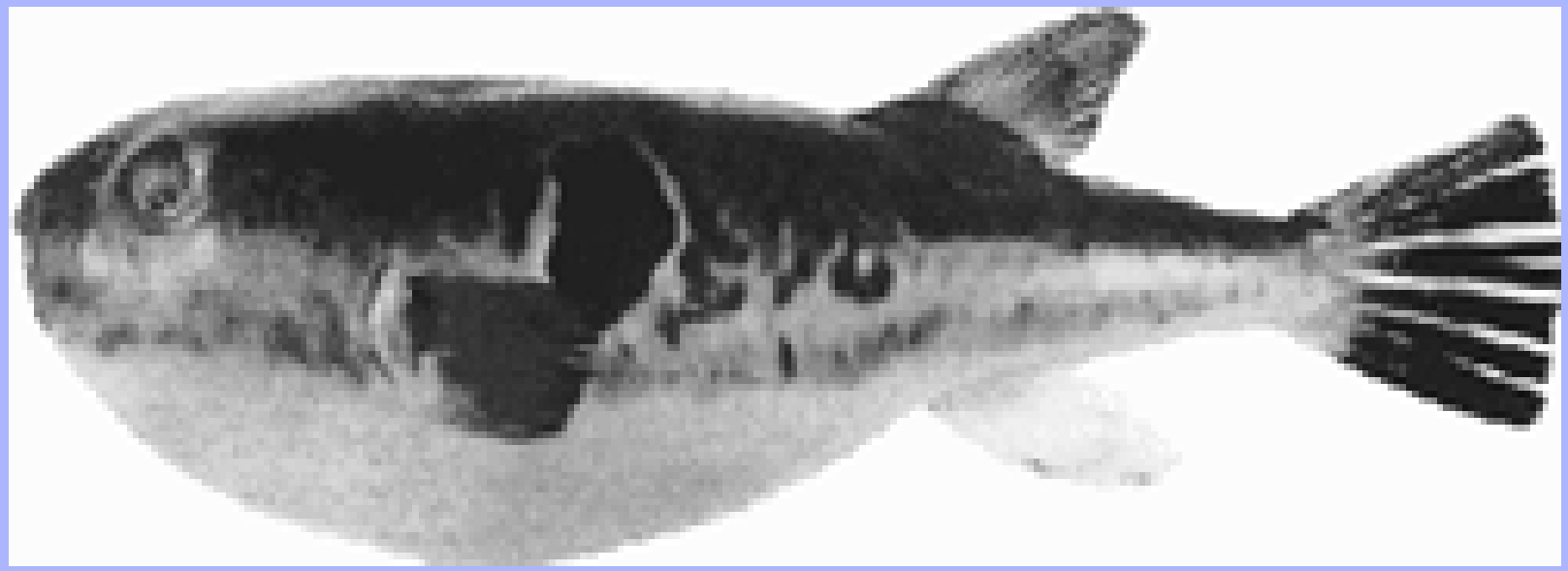
Canal Compiler Analysis of this MTA Code

```
|Loop 12 in find_all_scores at line 569 in region 9
 |   in parallel phase 2
 |   interleave scheduled
 |   dependences carried by: dot_time
 |   dependences carried by: rep_time
 |
 |Parallel Region 9 in find_all_scores
 |   multiple processor implementation
 |   requesting at least 115 streams
 |
```

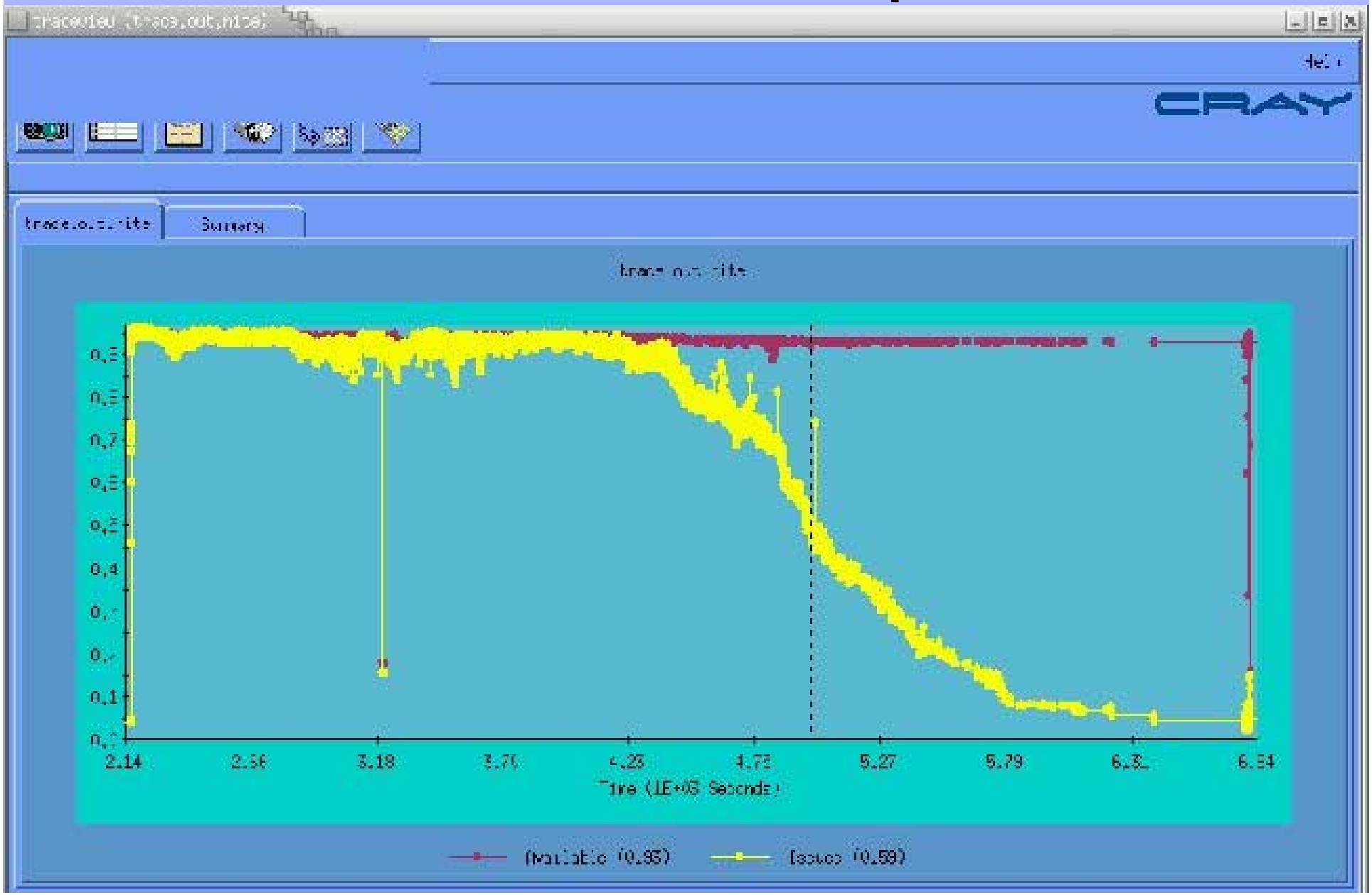
Performance of Cross-Match

- Parallelism is only limited by the number of reads in the input file and the number of available threads on the machine.
- We need a large data set to fully exploit the machine so…..

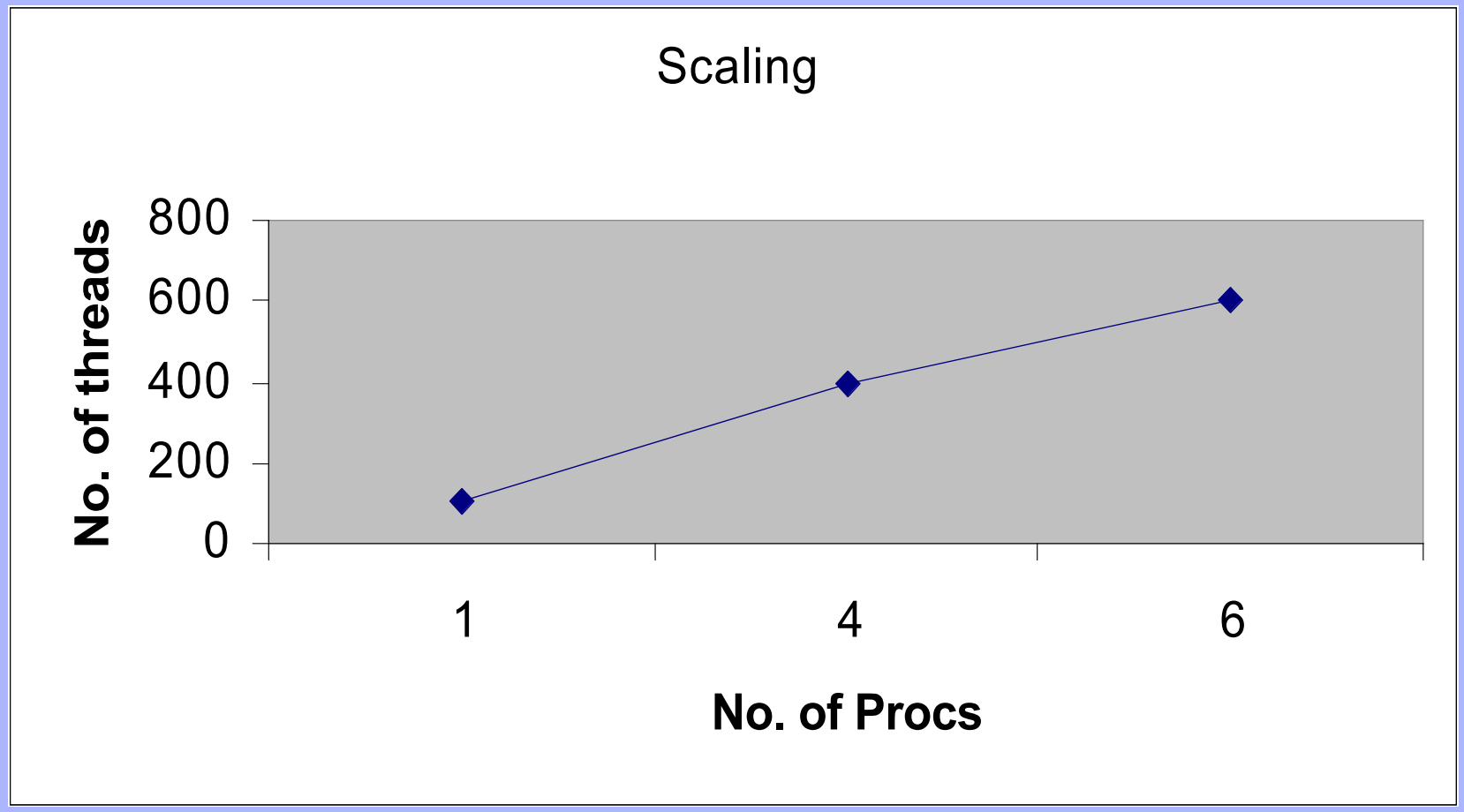
Name That Fish!



Traceview Output



Linear Scaling



Yeah, but how fast is it?

- With a 10MB (10,000 entries) input deck, the MTA (200MHz) does 50% of its swatting in 651 secs.
- An Origin 2000 (400MHz) does the same in 360 secs.
- However, this is still a fairly small input.

Conclusion

The enormous data sets associated with genomics and the inherently parallel nature of the processing provide an excellent opportunity for the MTA to show off its potential; it is starting to provide some promising results.