Top Ten Things To Try

When attempting to optimise an application

1. Profile, profile, profile

The most important thing in any optimisation is to understand the properties of the application.

Use profiling tools to understand what dominates runtime and where bottle necks exist.

- User functions?
- Libraries?
- MPI?
- Load imbalance?

Optimisation is an iterative process, target the longest running parts first.

Make sure your test case is representative of what your actual run

Know how to check your results are correct.

2. Adapt your environment

Many of the libraries and software on the Cray XE6 can be modified through the use of environment variables.

Significant amounts of MPI? Try adjusting the MPICH environment variables.

Lots of information available in:

man mpi
man crayftn
man libsci

http://docs.cray.com

http://www.hector.ac.uk/support/documentation/

3. Consider how you distribute data

Parallel performance can be strongly affected by the nature of the decomposition.

If your application has flexibility in its decomposition try experimenting with some different combinations.

Changing decompositions can affect the message sizes, vector lengths , cache usage.

Is the assumed wisdom about decompositions (especially for other HPC architectures) correct? It is often worth reassesing when running new models or when porting architectures.

4. Be careful with your bindings

With multi-core architectures with different NUMA regions placement of processes and threads on the node becomes important.

Binding and placement is controlled by the `aprun` command.

By default, each processor or thread is bound to an individual core, if you're spawning lots of threads this may not be what you want, try `-cc numa_node` or `-cc none`.

Perhaps experiment with using Interlagos in "core-pair" mode. Bind to every other processor:

```
aprun -n <nproc> -N <ntask> -d 2 <exe>
```

Consider under populating the node (cores are plentiful, perhaps you can afford not to use them), allows core specialisation.

5. Leverage the supplied libraries

The Cray environment provides a rich assortment of libraries. Much easier to use someone else's work.

Each library is built specifically for the target architecture and is included by the `ftn` and `cc` commands when the module is loaded.

Some of the libraries have been modified by Cray to improve their performance on the Cray, e.g. specifically targeting the network, optimising for iterative refinement.

The scientific libraries are threaded, if your code makes significant use of specialist calls then you can automatically go multithreaded by setting `OMP_NUM_THREADS`.

6. Consider using huge pages

Huge pages can optimise communication over the Gemini interface.

They can also improve serial performance by reducing misses in the TLB cache.

Adding support is very simple, just include any craype-hugepage module while compiling, the load an individual module at run time to set the default huge page size.

If you see a slow down when adding huge pages, please let us know.

7. Tailor your I/O

Match you Lustre striping settings to your application's I/O pattern.

Make sure you include as much parallelism as possible (open multiple files, use a parallel I/O layer like MPI-IO).

8. Time for different flags, or even compiler?

Adding optimisation when compiling can significantly improve the serial performance of an application.

Target files/routines which consume large amounts of I/O

Be aware that adding optimisation can affect the floating point results.

The advantage of using an X86 architecture is there are a variety of compilers available. Consider experimenting with other compilers available on the system.

What was optimal 12 months ago, may not be so today.

9. Remember the memory?

Make good use of caches, cache reuse is one of the most important optimisations on a modern HPC processor

Identify where there is poor cache utilisation (low reuse, hardware counters etc). Try and identify ways to improve the usage (process in smaller chunks, minimise the number of intermediate temporary arrays).

Remember that the first thread to “touch” or “initialise” a segment of memory will cause it to be bound to its local memory. Make sure arrays are initialised by the threads that will be using them most.

10. Minimise postage and packaging

Consider aggregating communications (especially collectives) into larger messages.

If your application can overlap communication and computation consider using the async comms threads to improve overlap.

Otherwise, consider using post MPI_Isend and using MPI_Recv in place of MPI_IRecv, MPI_Waitall

Consider using other parallel paradigms to avoid packing and unpacking data (e.g. Fortran coarrays, transfer data directly into/from arrays).

All parallel paradigms (MPI, SHMEM, UPC and Fortran Coarrays can be used in the same program on the Cray XE6).