A further test case using much larger orders (100..1000) is planned to determine the scalability of the threaded solution.

6.2 Some Threading Rules

In developing the threaded solution to this problem three axioms became apparent. The first is "***understand the problem***", it is crucial to understand what is trying to be accomplished. It is important to have a complete understanding of the formal solution and its practical implementation. In this case study the formal mathematical solution provided the basis for the practical solution and also indicated where the performance bottlenecks would occur. While this may not be possible in all cases, for CPU intensive problems this should be true most of the time.

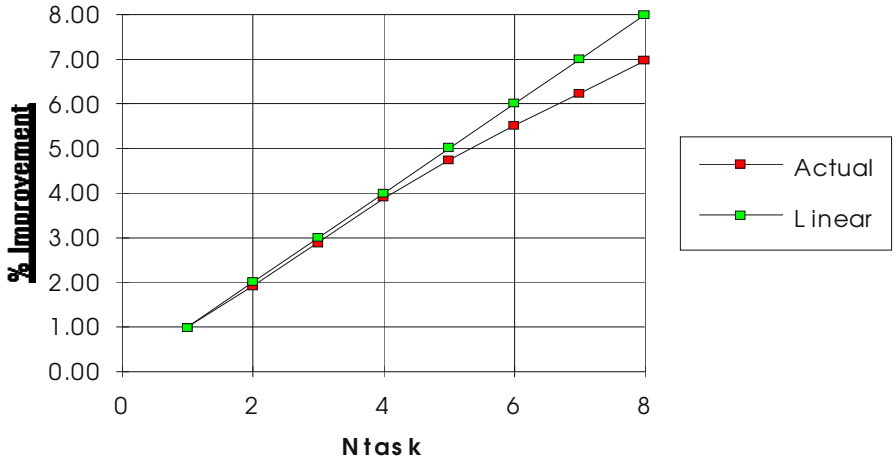
The second axiom is "***understand data generation and flow***". In using threads data locking is vital to ensure data integrity. Therefore, having a clear understanding of where and how data is created and used is essential. In analyzing the data generation of this case study it was determined that a large amount of concurrency was possible due to the independence of data generation. The understanding of data flow also allowed for concurrency.

The final axiom is "***trade-off effort for performance improvement***", while greater performance gains could perhaps be realized with more investigation and effort the gains realized in this case were demonstrated to meet the goals.

7.0 Acknowledgments

This work was carried out with the support of my manager Dean Kemp, who allowed the time to develop the solution and this document. The Benchmark Group (STE) of Sun Microsystems under the direction of Douglas Hill provided access to their very busy test system on short notice. Craig Sparkman (of STE) ran the benchmarks and provided compiler giggery pokery for optimal performance.

FIGURE 3. Number of CPUs and Improvement



The table and figure show that the usage is almost linear up to four. At this point the performance decays slightly until at 8, one of the CPUs is not being utilized. This decay is likely a result of the usage of the matrix tasking function. When the program starts it computes (in parallel) one vector and two matrices. When *Ntask* is set to four, this will create 9 active threads (one for the vector and four for each of the two matrices). This achieves almost linear performance with little loss due to thread contention on the 8 CPUs. However, when *Ntask* is set to eight then 17 threads are created. The result is more contention for the available (8) CPUs and more involvement required by both the kernel and threads library.

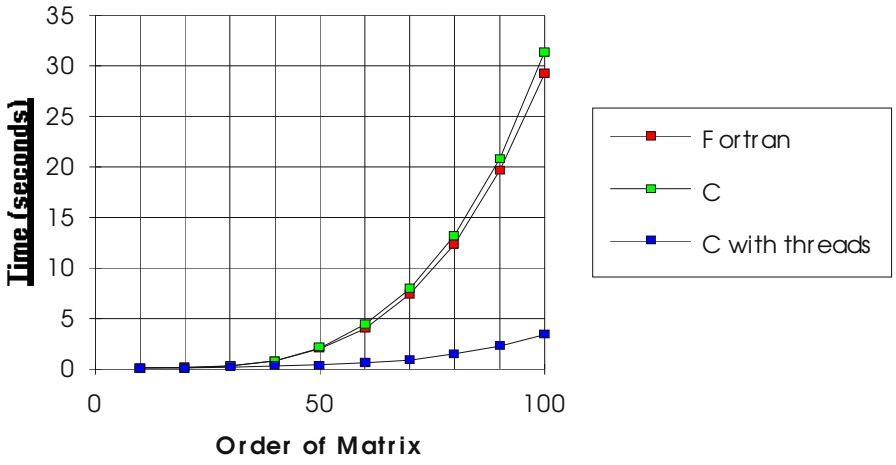
6.0 Conclusions

6.1 Threading Effort and Benefit

The time required to introduce threads to the C source was approximately 3 man-days. This included identifying where to thread, developing a matrix tasker to generate threads within matrix operations and writing the code. The source file was modified such that both the threaded and non-threaded were included in the same file. The original, single threaded program ran X lines, after the introduction of threads the file was Y lines. In comparison, the original FORTRAN program ran Z lines. However, the FORTRAN code contained virtually no whitespace or comments and contained a very simple linear equation solver (W lines of code).

The performance improvement varies greatly. On small order problems the difference between any of the solutions is negligible. However, for large order problems (50 or more) the threaded code starts at 5.3x faster and increases to 9.2x times faster for the 100 order case.

FIGURE 2. Order of Solution and Time



The figure clearly demonstrates the performance improvement of that can be achieved using threads on an multi-processor system. The C and FORTRAN cases are approximately equal and exhibit the aforementioned $O(N^4)$ behavior.

The second table compares the effect of the *Ntask* parameter on the execution time of the threaded solution.

TABLE 2. Number of CPUs and threaded solutions.

Ntask	C with threads (seconds)	CPU Usage
1	23.7	1.0
2	12.4	1.9
3	8.2	2.9
4	6.1	3.9
5	5.0	4.7
6	4.3	5.5
7	3.8	6.3
8	3.4	7.0

CPU usage is defined as $Usage_{Ntask} = time_1 / time_{Ntask}$

The following figure compares *Ntask* with CPU usage.

In the implementation of the second option a matrix tasking function was developed. The function splits a matrix (by rows) evenly into *Ntask* jobs, creates *Ntask* threads and waits until the threads finish. The value of *Ntask* is an argument to the tasking function, a value of zero will cause *Ntask* to be set to the number of on-line CPUs. Most of the time in creating the threaded version of the program was spent developing the matrix tasker. Once this was accomplished the original source was easily converted to use the tasker.

5.3 Benchmark Results

All tests were conducted using the Solaris 2.3 operating system. The tests were carried out on an SS1000 with 8 - 50 Mhz CPUs, 512 MB of RAM and 4 - 535 Mb drives The first table compares the order of matrix and various language solutions. The original FORTRAN is the original source code that ran on the Honeywell 6000. The original C is a conversion to C based on the original FORTRAN with the addition of a more robust equation solver. Threaded C is based on the original C source with source code changes to use the threads library.

The first table compares all solution types against the order of the solution. These tests were conducted with *Ntask* set to the number of on-line CPUs.

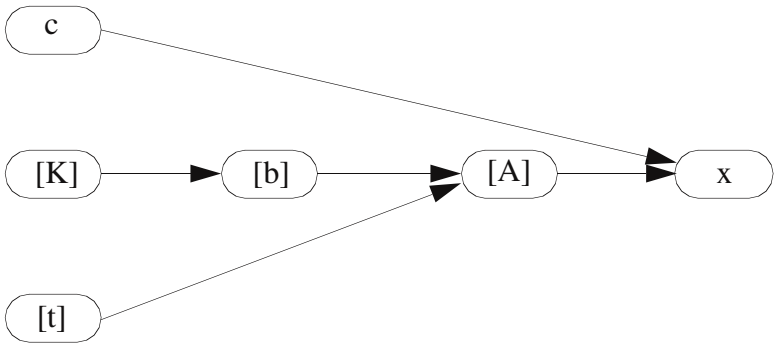
TABLE 1. Order of equations and solution type.

	FORTRAN	C		
Order	**Original**	**Original**	**Threaded**	**Improvement**
10	0.1	0.1	0.1	1.0
20	0.2	0.1	0.1	1.0
30	0.3	0.3	0.2	1.5
40	0.8	0.8	0.3	2.7
50	2.0	2.1	0.4	5.3
60	4.0	4.4	0.6	7.3
70	7.4	7.9	0.9	8.8
80	12.3	13.2	1.5	8.8
90	19.7	20.8	2.3	9.0
100	29.2	31.4	3.4	9.2

All times in seconds. Improvement is defined as $i_N = time_{(C, N)} / time_{(Threads, N)}$

The table is presented graphically below.

FIGURE 1. Data Dependency Graph



Examining the above figure finds that among the six data items there are only three synchronization points[2]. This indicates that a high degree inter-data concurrency is possible. Further examination of the six data items finds four are matrices (K, t, b, A) where the computation of the individual elements are independent. This will allow for a high degree of intra-data concurrency.

With the above facts, three options were proposed.

1. Create one thread to compute each of the six datum that are required.
2. Create N threads to compute the six datum, where N is greater than six. This involves threading some or all of the matrix generation steps.
3. Create a threaded data pipeline to compute individual matrix and vector elements.

Examining each of the options we find that:

- Option 1 is the fastest to implement and has the least impact on the source code, but it ignores the independence of the generation of the individual matrix elements.
- Option 2 exploits the independence of the matrix generation steps, but requires more effort to implement. The effect on the source code will not be as minimal as option 1, however, the performance improvement will likely be greater.
- Option 3 makes use of the independence of the matrix generation steps, but would have a major impact on the source code. Implementing this option would require a major time investment to create the threaded data pipeline along with a synchronization model for the data passing.

The second option was chosen as the model to proceed with when the goals were examined. This should produce the greatest benefit with the least source code impact and time investment.

2. Defining a synchronization point as an oval on the graph where one or more arrows meet at a datum.

2. Generate $K(x, y)$ matrix. This matrix is the evaluation of the function $K(x, y)$ at the required points for the computation of Chebyshev coefficients to the function. Order: $O(N^2)$
3. Compute Chebyshev coefficients of $K(x, y)$, the b matrix. Order: $O(N^4)$
4. Compute the t matrix. Order: $O(N^2)$
5. Compute the product of the b and t matrices and add the identity matrix. Order: $O(N^3)$
6. Solve for the coefficients of $f(x)$. Order: $O(N^2)$
7. Evaluate $f(x)$ at M points in the interval. Order: $O(N)$

Initial investigations into the time required to compute a solution for a given degree of approximation found that a fourth degree polynomial approximation provided a "best fit." This is in agreement with the evaluation presented above.

5.0 Threaded Solution

5.1 Goals

Before commencing on the project the following goals were recognized.

1. Minimize the impact on the existing program at the source level. The final source file should contain both the threaded and non-threaded version of the program with a minimum of duplicate code. The POSIX feature test macro _REENTRANT can be used to selectively include portions of code.
2. The time for conversion should be small in comparison to the time required to code the original program. The original program consumed approximately five man-weeks of time to develop and test. This time included the development of the formal solution.
3. Maximize performance improvement but not at the expense of the above two goals. This goal was difficult to quantify, it is best stated as "enough threading, but not too much."

5.2 Solution Options

By examining the formal solution presented above the following data dependency graph was generated.

with x_k given by (EQ 6) and y_l given by the same equation with appropriate substitution. The value of *P* is best stated as: equal to one (1) on the corners, two (2) elsewhere on the periphery and four (4) everywhere in the matrix.

Now, by substitution of equations (EQ 4) and (EQ 10) into (EQ 9) results in following expression of the integral equation.

$$\sum a_r \times T_r(x) + \lambda \times \int \left[\left(\sum\sum b_{rs} \times T_r(x) \times T_s(y) \right) \times \left(\sum a_s \times T_s(y) \right) \right] dy = \sum c_r \times T_r(x) \qquad \text{(EQ 12)}$$

By expanding the integral and collecting terms, the integration can be re-written as a matrix multiplication of the $\boldsymbol{b}$ matrix of Chebyshev coefficients to $K(x, y)$ and a $\boldsymbol{t}$ matrix with elements given by

$$t_{ij} = \int T_i(x) \times T_j(x) dx \qquad \text{(EQ 13)}$$

At this point (EQ 12) can be written as a system of linear equations

$$\boldsymbol{I} \cdot \vec{x} + \lambda \times \boldsymbol{b} \cdot \boldsymbol{t} \cdot \vec{x} = \vec{c} \qquad \text{(EQ 14)}$$

and rearranged to the familiar $\boldsymbol{A} \cdot \vec{x} = \vec{b}$ form

$$(\boldsymbol{I} + \lambda \times \boldsymbol{b} \cdot \boldsymbol{t}) \cdot \vec{x} = \vec{c} \qquad \text{(EQ 15)}$$

The solution of which can be provided by any number methods to solve a system of linear equations.

4.0 Original Solution

4.1 Procedure

The original solution to the problem identified a number of steps to be performed. These steps are based on the formal solution presented above. The solution requires the user to provide six parameters, which are:

1. Range of definition: $[a, b]$ and scalar λ.
2. Functions: $K(x, y)$ and $g(x)$
3. Degree of approximation: N.
4. Number of points to evaluate $f(x)$: M.

The procedure to solve the equation is presented below, each step includes an estimate of the order of computation to perform the step.

1. Compute Chebyshev coefficients of $g(x)$. Order: $O(N) + O(N^2)$

$$f(x) = \sum_{r=0}^{N} a_r \times T_n(x), -1 \le x \le 1 \quad \text{(EQ 4)}$$

where the coefficients a_r are:

$$a_r = \frac{2}{N} \sum_{k=0}^{N} f(x_k) \times \cos\left(\frac{k\pi r}{N}\right) \quad \text{(EQ 5)}$$

and the x_k values are given by:

$$x_k = \cos\left(\frac{k\pi}{N}\right) \quad \text{(EQ 6)}$$

One important property that is used for the generation of $K(x, y)$ Chebyshev coefficients is

$$T_n(x_k) = T_k(x_n) \quad \text{(EQ 7)}$$

where the x_k values are given by (EQ 6).

3.3 Approximate Solutions

The solution presented here is the condensed version of a longer and more formal presentation that appeared in the original research work.

Any finite interval $a \le x \le b$ can be converted to the interval $-1 \le y \le 1$ using the transformation:

$$y = \frac{2x - (b + a)}{b - a} \quad \text{(EQ 8)}$$

Then by substitution of (EQ 8) into (EQ 1) and re-writing in terms of x yields

$$f(x) + \lambda \times \int_{-1}^{1} K(x, y) \times f(y) dy = g(x), -1 \le x \le 1 \quad \text{(EQ 9)}$$

An expansion to $f(x)$ is already provided by (EQ 4). A Chebyshev approximation to $K(x, y)$ is given by

$$K(x, y) = \sum_{m=0}^{N} \sum_{n=0}^{N} b_{ij} \times T_m(x) \times T_n(y) \quad \text{(EQ 10)}$$

where the b_{ij} can be shown to be given by

$$b_{ij} = \left(\frac{P}{N^2}\right) \times \sum_{k=0}^{N} \sum_{l=0}^{N} K(x_k, y_l) \times T_i(x_k) \times T_j(y_l) \quad \text{(EQ 11)}$$

2.1 Integral Equations

An integral equation is analogous to a differential equation. It involves finding a function that satisfies an expressed relationship between known functions, their integrals and an unknown function bounded by a range of definition. Integral equations find application in many fields of engineering including vibrations, solid mechanics, heat and fluid flow.

This case study concerns itself with the solution of one type of integral equation: Fredholm integral equations of the second kind. These equations have the form:

$$f(x) + \lambda \times \int_a^b K(x, y) \times f(y)dy \; = \; g(x), \, a \le x \le b \qquad \text{(EQ 1)}$$

where the functions $K(x, y)$ and $g(x)$ are specified along with the range $a \le x \le b$ and scalar λ. The function $f(x)$ is the required solution to the equation.

In the field of solid mechanics the function $K(x, y)$ is a regular function that describes the contact stresses and geometry, $g(x)$ is proportional to the displacement and $f(x)$ is proportional to the unknown contact pressure.

3.0 Formal Solution

3.1 Analytical Solutions

As is the case with most "real-world" problems, finding an analytical solution is seldom possible. Most problems are solved using some approximation method that can be demonstrated to be sufficiently accurate over an interval. Fredholm integral equations are no exception and several different techniques can and have been employed.

3.2 Chebyshev Polynomials

The procedure developed here uses Chebyshev polynomials to find an approximate solution to the problem.[1] Chebyshev polynomials are one of a number of orthogonal polynomials types. The following equations define Chebyshev polynomials and some of the properties associated with them. The first two equations define the first kind Chebyshev polynomial.

$$T_n(x) \; = \; \cos\,(n\theta) \qquad \text{(EQ 2)}$$

$$\cos\theta \; = \; x, -1 \le x \le 1 \qquad \text{(EQ 3)}$$

Chebyshev polynomials can be used to approximate a function $f(x)$ using:

1. Under some circumstances the solution will be exact.

Threads Case Study #2

Richard Marejka
Solaris 2 Migration Support Centre
Sun Microsystems, Inc

1.0 Introduction

1.1 Threads History

Solaris 2 is the first operating system offering from Sun Microsystems that includes a symmetric, multi-processing kernel executing on Sun's multi-processor hardware. While kernel based threads have been available to operating system adventurers since Solaris 2.0, the user-level threads (ULT) library was not included in the initial Solaris release. The first release of the user-level threads library was as an Early Access package for Solaris 2.1 in September 1992. With the release of Solaris 2.2 in June 1993 the ULT library has become a bundled and integral component of Solaris.

1.2 Case Study Approach

This will (hopefully) be the first in a series of case studies that will demonstrate the application of threads to a number of computing problems.A case study approach allows the exploration of a specific class of problem and possible solutions employing threads. There are three benefits from this approach, they are:

- Measurement of the effort required to write or convert an application.
- Measurement of the performance improvement.
- Identification of techniques that can be applied for writing or converting other programs.

As with any decision the case study approach is not perfect. The primary fault is that it examines a specific problem and the inductive reasoning required for application to the class of problems as a whole is not always obvious.

2.0 Problem Description

This case study examines the solution of a problem from the fields of mathematics and engineering that is compute intensive. The original work was performed while the author was a research assistant in the Faculty of Engineering at the University of Waterloo. The original program was written in FORTRAN 77 and ran on a Honeywell 6000 series mainframe.