Malaysian Journal of Computer Science, Vol. 13 No. 1, June 2000, pp. 1-11

IMPLEMENTATION OF PARALLEL BOUNDARY INTEGRAL METHOD ON SPHERICAL BUBBLE DYNAMICS USING SHARED MEMORY COMPUTER

Rozita Johari
Mohd Yazid Md Saman
Mohamed Othman
Jabatan Sains Komputer
Jabatan Teknologi Komunikasi dan Rangkaian
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Putra Malaysia
43400 UPM Serdang
Selangor, Malaysia
email: rozita@fsas.upm.edu.my
yazid@fsas.upm.edu.my

Bachok Taib
Jabatan Matematik
Fakulti Sains dan Pengajian Alam Sekitar
Universiti Putra Malaysia
43400 UPM Serdang
Selangor, Malaysia
email: bachok@fsas.upm.edu.my

ABSTRACT

The boundary integral method is employed to model the dynamic behavior of the 3D spherical bubble. It has been solved on the Sequent Symmetry S5000 SE30 computer to better understand the opportunities and challenges the parallel processing presents. Analyses of the parallel performance of the approximation to the potential at certain external points as well as the normal derivatives of the potential on the surface of the bubble were generated using linear representations of the surface and the functions. In these calculations, 4, 6 and 8 Gauss points were used in the integration on 4, 8, 16 32 and 64 segments. Results from this study demonstrate that parallel computing greatly conserves the computational effort and is shown to be an effective tool for several problems related to bubble dynamics.

Keywords: ***Boundary integral method, Shared memory, Bubble dynamics, Parallel computing***

1.0 INTRODUCTION

The *Boundary Integral Method (BIM)* which sometimes referred to as the *Boundary Element Method (BEM)* is a well establish technique for the solution of problems in engineering and applied science as described by [21] and [18]. The BIM is a technique which often presents important advantages over domain type solutions since it provides a great economy in computational efforts by discretizing only the boundary of the domains. Consequently, much smaller systems of equations are to be solved. However for complex geometry, as in the three-dimensional case, dense meshes are required so that quite a large system of equations still remains, which make the solving step slow.

Development of parallel computer has received considerable attention by users of BIM. As described by [5], there are essentially three phases in BIM.

(i) The matrix set-up phase
(ii) The solution of linear equation phase
(iii) The calculation of external points phase.

The first parallel implementation was described in the literature by [13]. Symm's implementation, on the ICL DAP, comprised an indirect approach with constant elements for the solution of the Dirichlet problem in a circle. Since then there has been a considerable increase in the interest in the use of parallel architectures.

Various authors have described parallel computations only for certain aspects of the boundary integral method. In the early attempts, most workers concentrated on the linear equation solution phase. Parallel solution of the system of equations in the area of micro hydrodynamics was done by [9]. Complete fine-grained implementation, in which all phases exploit the parallelism, are described by [2], [3] and [4], who considered a variety of linear and quadratic element implementations of potential problems on the ICL DAP. The potential problem, applied to problems with a free surface, has been considered by [10] using a network of Sun workstations run in parallel using the Parallel Virtual Machine (PVM) parallel toolkit. A parallel implementation of a quadratic element approach to the solution of axis-symmetric elastostatic problems is given by [12].

The last decade has seen a great resurgence of bubble research using boundary integral method due to the growing realization of the importance of bubbles and bubble phenomena in science and technology as described by [8], [11], [14] and [15]. However none has been discussed on the parallel implementation of the bubble dynamics using the boundary integral method on a shared memory multiprocessor system. The goal of this paper is to analyze the parallel performance of the 3D spherical bubble using boundary integral method on a shared memory computer. This work was carried out on a Sequent Symmetry S5000 SE30 computer with ten processors using C language with parallel library. Section 2 gives the overview architecture of the Sequent SE30, which is a

MIMD multiprocessor system. Section 3 discusses the Boundary Integral Method and the spherical bubble using boundary integral method. Section 4 discusses the parallel algorithm of the spherical bubble and deals with it performance analysis. Section 5 provides concluding remarks.

2.0 MIMD ARCHITECTURE – SEQUENT SE30

Sequent Symmetry S5000 SE30 (also known as Sequent SE30) is a MIMD shared memory multiprocessor system. The shared memory machine, sometimes called the tightly-coupled system, has a set of processing elements (PEs) and a pool of memory available to all processors ([24], [17], [22] and [16]). The processors have access to a large global random access memory of which they have the same view. The software processes, executing on different processors, co-ordinate their activities by reading and modifying data value in the shared-memory. The co-ordination is achieved via different mechanisms that synchronize attempts to access the shared data. Processors are provided with a small fast local memory, in the form of data registers or cache. Access to the global memory is either via a high-speed bus or a switching network. Fig. 2.1 shows a simplified diagram of a shared memory parallel computer.

As described in [23], Sequent SE30 is a multiprocessor system comprising from two to ten 66 or 100 MHz Intel Pentium microprocessors on a dual-processor circuit board. Each Pentium microprocessors is coupled with a high-performance, two-megabyte secondary cache memory to support the CPU's 8-kilobyte on-chip data and instruction caches. System memory (RAM) is expandable to 3.5 gigabytes when the 100-MHz processors are used exclusively.

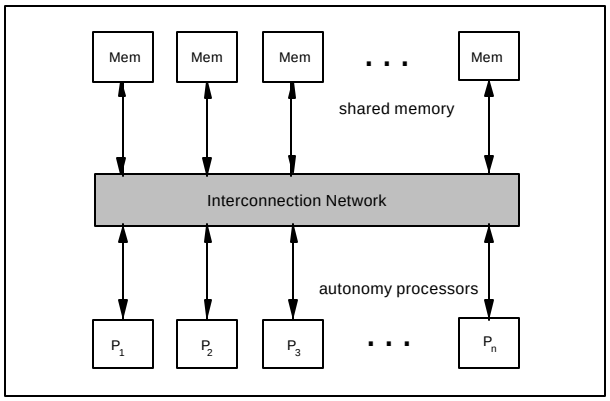


Fig. 2.1: A typical shared memory architecture

Disk storage is expandable to 940 gigabytes using the QCIC-E disk subsystem. The I/O capabilities of the SE30 are VMEbus-based, supporting up to 176 direct asynchronous serial connections, 32 high-speed synchronous connections, or 11 Ethernet connections. Fig. 2.2 illustrates the interconnections of the components of Sequent SE30 system.

The Sequent Highly Scalable Bus (HSB) is the primary communication path for the processors, memory boards, and I/O subsystems. This bus operates at both 10 MHz (for address and request packets) and at 30 MHz (for data packets) and provides a 64-bit data path multiplexed with a 32-bit address path. The HSB can achieve a peak data transfer rate of 240 megabytes per second. The system backplane also includes a 1-bit serial bus called the System Link and Interrupt Controller (SLIC) bus. The SLIC bus is used to exchange low-level data packets between the CPUs and other subsystems.

Sequent SE30 processor board contains two complete CPU systems. Each CPU system consists of an Intel Pentium CPU, a high-performance secondary cache memory to supplement the CPU's onboard cache memory, and support circuitry for the interface to the system bus.

The Sequent SE30 High Density Memory Controller board, installed in the HSB card cage, contains 64, 256, or 512 megabytes of system memory.

Sequent SE30 systems run DYNIX/ptx, Sequent's POSIX-compliant implementation of the UNIX System V operating system, and are binary-compatible with all Symmetry computer systems running DYNIX/ptx.

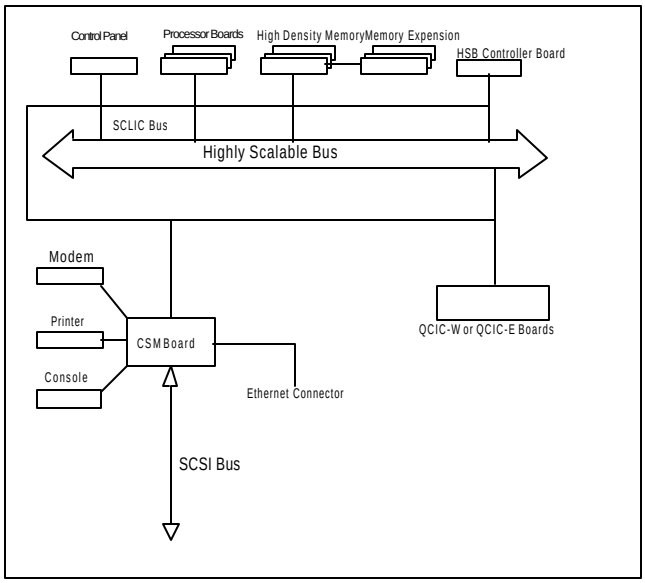


Fig. 2.2: Sequent Symmetry S5000 SE30 architecture

3.0 BOUNDARY INTEGRAL METHOD

3.1 Formulation

To illustrate the boundary integral formulation, consider Laplace equation $\nabla^2 f = 0$ in some domain Ω. The starting point of an equivalent integral formulation is to consider the appropriate fundamental solution for the Laplace equation. The boundary integral method is based

on Green's formula, which can reformulate the potential problem as the solution of a Fredholm integral equation. According to [14], for any sufficiently smooth function f which satisfies the Laplace equation with a domain Ω having piecewise smooth surface S, Green's formula can be written as in [7] and [1].

$$(1)\quad c(p)f(p)+\int_S f(q)\frac{\partial}{\partial n}\left(\frac{1}{|p-q|}\right)dS=\int_S \frac{\partial}{\partial n}(f(q))\frac{1}{|p-q|}dS$$

where $p \in \Omega + S$, $q \in S$, $\partial/\partial n$ is the normal derivative outward from S, and

$$(2)\qquad c(p)=\begin{cases}4 & if \quad p\in\Omega\\ 2 & if \quad p\in S\end{cases}$$

Equation for either the potential f or the normal derivatives of the potential $\partial f/\partial n$ if the other is specified yielded by choosing point *p* on the surface S. Once both are known on S, equation (1) can be used to generate f at any interior point *p*.

3.2 Axis-symmetric Form of the Integrals

Using cylindrical polar coordinates with $p=(r_0,0,z_0)$ and $q=(r,\theta,z)$ respectively,

$$(3)\qquad \left|\frac{1}{p-q}\right|=\frac{1}{[(r\cos\theta-r_0)^2+(r\sin\theta)^2+(z-z_0)^2]^{\frac{1}{2}}}$$

$$=\frac{1}{[(r+r_0)^2+(z-z_0)^2-4rr_0\cos^2\frac{\theta}{2}]^{\frac{1}{2}}}$$

If the surface S is parameterized by the arc length variable ξ

$$(4)\qquad \int_S\frac{1}{|p-q|}dS=\int_0^1\frac{4r(x)[(\frac{dz}{dx})^2+(\frac{dr}{dx})^2]^{\frac{1}{2}}}{[(r(x)+r_0)^2+(z(x)-z_0)^2]^{\frac{1}{2}}}K(k)dx$$

and

$$(5)\qquad \int_S\frac{\partial}{\partial n}\frac{1}{|p-q|}dS=\int_0^1\frac{4r(x)}{[(r(x)+r_0)^2+(z(x)-z_0)^2]^{\frac{3}{2}}}dx$$

$$\{[\frac{dz}{dx}(r(x)+r_0)-\frac{dr}{dx}(z(x)-z_0)-\frac{2}{k^2(x)}\frac{dz}{dx}r_0]\frac{E(k)}{1-k^2(x)}+\frac{2}{k^2(x)}\frac{dz}{dx}r_0K(k)\}$$

where

$$(6)\qquad k^2(x)=\frac{4r(x)r_0}{(r(x)+r_0)^2+(z(x)-z_0)^2}$$

and K(k), E(k) are the complete elliptic integrals of the first and second kind. Approximation for these functions is available in [6] in the form

$$(7)\qquad K(k)=P(x)-Q(x)\ln x$$

$$E(k)=R(x)-S(x)\ln x,$$

where

$$(8)\qquad x=1-k^2(x),$$

and P, Q, R, and S are tabulated polynomials.

3.3 Surface Approximation

First of all, representations for the surface of the bubble, the potential and its derivatives on the surface need to be chosen. These choices can be independent, but as the movement of the surface is computed using the potential and its derivatives, both should be considered together and will be called functions. In the description below, a plane section through the axis of the symmetry of the bubble is taken, and rotational symmetry about the axis is understood.

Constant Approximation

For the constant segments considered here, the surface is assumed to be divided into N elements. The potential and its normal derivative constant over each element and equal to the value at the mid-element node.

$$(9)\qquad 2\pi f_i+\sum_{j=1}^{N}f_j\int_{S_j}\frac{\partial}{\partial n}\left|\frac{1}{p_i-q_j}\right|dS=\sum_{j=1}^{N}\frac{\partial}{\partial n}(f_j)\int_{S_j}\frac{1}{|p_i-q_j|}dS$$

We can write (9) in matrix form as in [1]

$$(10)\qquad 2\pi f_i+\sum_{j=1}^{N}\hat{H}_{ij}f_j=\sum_{j=1}^{N}G_{ij}\frac{\partial f}{\partial n}$$

Defining $H_{ij}=\hat{H}_{ij}+2\pi\delta_{ij}$ (10) may be written as

$$(11)\qquad Hf=G\frac{\partial f}{\partial n}$$

Linear Approximation

If a linear approximation is used then the bubble surface is replaced by a set of *N* linear segments S_j with f and

$\partial f/\partial n$ are assumed to be single valued at the end points of the linear segments. If the segment is parameterized by ξ in the range (0,1), we can define

$$M_1(x) = 1 - x \tag{12}$$

$$M_2(x) = x$$

and use the isoparametric approximations for both the surface and the functions. At the jth segment, S_j, the surface is defined by

$$r(x) = r_{j-1}M_1(x) + r_j M_2(x) \tag{13}$$

$$z(x) = z_{j-1}M_1(x) + z_j M_2(x)$$

With the values of f and $\partial f/\partial n$ on S_j defined by

$$f(x) = f_{j-1}M_1(x) + f_j M_2(x) \tag{14}$$

$$\frac{\partial f}{\partial n}(x) = \frac{\partial f_{j-1}}{\partial n} M_1(x) + \frac{\partial f_j}{\partial n} M_2(x)$$

Since the collocation points are moved to the end points of the interval, it will yield $N+1$ equations in the $N+1$ unknowns. The integral on each segment can be written

$$\int_{S_j} dS \frac{\partial f}{\partial n} \frac{1}{|p_i - q_j|} = b_{1ij} \frac{\partial f_{j-1}}{\partial n} + b_{2ij} \frac{\partial f_j}{\partial n} \tag{15}$$

where

$$b_{kij} = S_j \int_0^1 dx M_k(x) \int_0^{2p} dq \frac{1}{|p_i - q(x,q)|} \tag{16}$$

$$\int_{s_j} dS f \frac{\partial}{\partial n}\left(\frac{1}{|p_i - q_j|}\right) = a_{1ij} \frac{\partial f_{j-1}}{\partial n} + a_{2ij} \frac{\partial f_j}{\partial n}$$

where

$$a_{kij} = S_j \int_0^1 dx M_k(x) \int_0^{2p} dq \frac{\partial}{\partial n}\left(\frac{1}{|p_i - q(x,q)|}\right) \tag{17}$$

3.4 Numerical Integration

Green's Integral formula is solved numerically by writing equation (1) in the matrix form as in (11) where H and G are matrices and f and $\partial f/\partial n$ can be calculated by solving a set of linear equations. The evaluation of the elements of the matrices H and G is performed numerically. Normally Gauss Legendre quadrature is adequate, unless the collocation point P_i is within the segment S_j, or is one of its end points, in which case the integrand is singular and must be treated specially. The singular integrals are evaluated by subtracting a logarithmic term to remove the singularity, then using a quadrature scheme incorporating the logarithm to complete the integration. Further details of the numerical integration may be found in [14].

3.5 Numerical Test

Initially the spherical bubble is centered at *(r,z)=(0,0)* of radius 1.0 in an infinite medium. A uniform potential $f = 1$ is prescribed on the surface of the bubble. Approximation to the potential at certain external points are generated as well as the normal derivatives of the potential on the surface of the bubble using N= 4,8,16,32 and 64 segments. The exact solution of the potential is given by $f = \frac{1}{[r^2 + z^2]^{1/2}}$ and the exact normal derivative is given by $\frac{\partial f}{\partial n} = -1$.

4.0 PARALLEL IMPLEMENTATION

4.1 Loop Level Parallelism

Loops are a rich source of parallelism. Executing loop iteration in parallel on a multiprocessor system will improve the execution of a program. The iteration of a loop are considered as independent tasks and are scheduled for execution on a shared memory multiprocessor system using some loop scheduling strategy. There are two main categories of algorithm to schedule loop iteration: Static and dynamic. Static scheduling requires no communication between processes and is generally used when you know that the computing time is approximately the same for each iteration of your loop. Static scheduling assigns iteration to the processors at compile-time. Each processor knows exactly which iterations it should execute before the program is invoked and, therefore, there is no scheduling overhead. For example, processor 0 executes iterations 1, P+1, 2P+1,…, processor 1 executes iterations 2, P+2, 2P+2,…, and so on, where P is the number of processors. The main disadvantage of static scheduling is load imbalance as described by [19] which causes some processors to remain idle while others are busy. This imbalance can cause by difference in iteration execution times, or by differences in the number of iterations each processor executes.

Dynamic scheduling attempts to reduce load imbalance by having idle processors assign iterations to themselves at run-time. However, dynamic scheduling creates more communication overhead than static scheduling because all the processes must access a single shared task queue.

In this paper, the static scheduling is used based on three reasons. First, static scheduling results in lower execution times than dynamic scheduling. Second, static scheduling can allow the generation of only one process per processor,

reducing process creation, synchronization, and termination overhead. Third, static scheduling can be used to predict the speedup that can be achieved by a particular parallel algorithm on a target machine, assuming no preemption of processes occurs.

4.2 Parallel Algorithm

In this paper, we are not going to address the problem associated with the equation solution phase since the solutions of linear equation using Gauss elimination with partial pivoting have been widely discussed in the literature ([20], [24], [25], [26], [27]). This paper will concentrate on the matrix set up phase and the calculation of the external point phase. The parallel algorithm for the 3D spherical bubble is shown in Fig. 4.2.1.

```
0:  P3DSB Algorithm
1:  {
2:  Input data (# of boundary element, # of external
    points, the x-y coordinates of the external
    points,initial condition of the spherical bubble
    centered at (r,z)=(0,0) with radius 1.0, theta=π/(#of
    boundary elements)).
3:  Shared memory allocation sets all matrices to
    address newly allocated shared matrix.
4:  Set the number of processors to run all the three
    phases in parallel {
5:          m_fork (setup_matrix).
6:          m_fork(solve_linear_equation).
7:          m_fork(compute_external_points).
8:          m_kill_procs();
9:  }
```

Fig. 4.2.1: Parallel Algorithm for 3D Spherical Bubble

The system equation set-up phase is a fine-grained process comprising a set of three nested loops. The inner loop, over the Gauss points, contains the straightforward calculation of the contribution to the coefficients H_{ij} and G_{ij}. The intermediate loop is over the target elements, [*j*], and the outer loop is over the base nodes, the collocation points [*i*]. When implementing on Sequent SE30 shared memory processors, the two outer loops are affected simultaneously in parallel and the inner loop only is performed sequentially. The parallel algorithm for the matrix set up phase is shown in Fig. 4.2.2.

The calculation of the external point phase is comprising of three nested loops. The inner loop, over the Gauss points is a sequential calculation of the approximation to the potential. The intermediate loop is over the target elements, and the outer loop is over the external points. Both the outer loops are affected simultaneously in parallel. The parallel algorithm for the calculation of the external point phase is shown in Fig. 4.2.3.

```
5.0: PSETUP Algorithm
5.1: {
/*Assembling G and H and form the system ax=b */
     id=m_get_myid(); /*get id of each processor */
   np=m_get_numprocs();/*get number of processors*/
5.2: for (i=id+1 to number of base node increment for
          every processors{ /*in parallel*/
       if (i==1 || i==n) {/*collocation points on the axis */
          k=n-2;
5.3:   for (js=1; js <=k; ++js){ /* parallel over the target
          element*/
          j=i+js;
        if (j>=n) j=j-nm1;
        if (i==n) j=j-1;
        inlul(); /* sequential over the Gauss points */
       /* formation of coefficient H_ij and G_ij */
}}
else{
k=n-3;
5.4: for (jj=1;jj<=k;++jj){/*parallel over the target
                          element */
    j=i+jj;
    if (j>=n) j=j-nm1;
    inte();  /* sequential over the Gauss points  */
    /* formation of coefficient H_ij and G_ij */
}}}
m_sync();  /*synchronization of all processes */

5.5: for (i=id+1 to number of base node increment for
          every processors{ /*in parallel*/
     if (i==1) {
     j=i;          /* parallel over the target elements   */
     inlul(); /* sequential over the Gauss points   */
     /* formation of coefficient H_ij and G_ij */
       continue;
    }
    if (i==n) {
    j=n-1;   /*   parallel over the target elements  */
    inlul(); /* sequential over the Gauss points  */
    /* formation of coefficient H_ij and G_ij */
    continue;
     }
     j=i-1;  /*  parallel over the target element   */
5.6: for (jk=1; jk<=2; ++jk) {
     if (jk == 1) {
     inlor()    /*  sequential over the Gauss points  */
     }
    else{
    inlol();   /*   sequential over the Gauss points  */
    }
      j=j+1;
   }
}

m_sync();  /*synchronization of all processes */

5.7: for (i=id+1 to number of base node
       increment for every processors{ /*in parallel*/
          h[i][i]=4.0*pi-h[i][i];
          dfi[i]=0.0;
            for (j=1; j<=n; ++j)
                dfi[i]=dfi[i]+h[i][j]*fi[j];
     }
5.8: }
```

Fig. 4.2.2: Parallel Algorithm of Setup Phase

```
7.0: PEXTPOINT Algorithm
7.1: {
7.2: /*Calculation of the external point phase in parallel */
7.3:  For(k=id+1 to number of points increment for every
          number of processors) {
          Sol_of_potential[k]=0.0;
7.4:   For(j=1 to number of segments) { /* in parallel */
          inte(); /*sequential over the Gauss points */
          calculation of potential(sol[k])
       }
7.5       sol[k]=sol[k]/(4.0*pi);
     }
7.6: }
```

Fig. 4.2.3: Parallel Algorithm for External Point Phase

4.3 Performance Analysis

The combination of parallel algorithm and parallel architecture is known as a parallel system. *Qualitative analysis* is one way to evaluate the performance of parallel system. There are various metrics that can be used to evaluate the performance of parallel systems under the *qualitative analysis*. The metrics are *execution time, speedup, and efficiency.*

The *execution time* of a serial program (denoted T_s) is the time elapsed between the beginning and the end of its execution on a single processor. The *execution time* of a parallel program on *p* processors (denoted T_p) is the time that elapses from when the first processor starts executing on the problem to when the last processor completes execution.

Speedup is the measure of relative benefits of paralleling a given application over sequential implementation. There are several ways of defining speedup. [20] defined the speed-up ratio on *p* processors as below:

$$S_p = T_0 / T_p$$

where T_0 is the time for the fastest serial algorithm for a given problem. However, the speed-up ratio that we use is known as the algorithmic speed-up ratio on *p* processors and it is defined as below:

$$S_p = T_1 / T_p$$

T_1 is the time taken by the parallel implementation executing on a single processor. The above algorithm speed-up ratio is the ratio of the time taken to solve a problem on a single processor to the time required solving the same problem on a parallel computer with p identical processors. Normally, T_1 is always exceeding the time taken by T_0.

Efficiency (E) is a measure of the fraction of time that processors speed doing useful work. It is defined as the ratio of speedup to the number of processors. It can be defined as:

$$E = \frac{S}{P}$$

This measure provides an indication of the effective of the *p* processors relative to the given algorithm. A value of *E* approximately equal to 1, for some *p*, indicates that algorithm A runs approximately *p* times faster using *p* processors than it does with one processor.

4.4 Experimental Result

The performance of the algorithm was measured by implementing both the serial and parallel versions in C with parallel library. The algorithm was run on the Sequent SE30 MIMD shared memory computer with ten processors.

First we will look at the percentage of errors of the potential f and the normal derivatives $\partial f/\partial n$. Errors are produced by subtracting the numerical solution from the exact solution. From Table 4.4.1, we can see that the percentage of error f and $\partial f/\partial n$ is decreasing as the number of segments is increasing. We can say that the numerical result is improving by using larger number of segments. However, this will lead to a longer execution time due to bigger size of matrices. This is also true for different numbers of Gauss points, where as we increase the number of Gauss points, the execution time is increasing.

Table 4.4.1: Sequential Results

4 Gauss Points			
Number of Segments	Sequential Time(Seconds)	% error in f	% error in $\partial f/\partial n$
4	0.004136	2.46	13.3672
8	0.011908	0.629	3.1255
16	0.040232	0.159	0.7715
32	0.147489	0.04	0.1957
64	0.579267	0.01	0.0527
6 Gauss Points			
Number of Segments	Sequential Time(Seconds)	% error in f	% error in $\partial f/\partial n$
4	0.005663	2.46	13.3627
8	0.017188	0.629	3.1205
16	0.058199	0.159	0.7664
32	0.213420	0.04	0.1907
64	0.826594	0.01	0.0477
8 Gauss Points			
Number of Segments	Sequential Time(Seconds)	% error in f	% error in $\partial f/\partial n$
4	0.007551	2.46	13.3626
8	0.022519	0.629	3.1205
16	0.076445	0.159	0.7664
32	0.278622	0.04	0.1906
64	1.075892	0.01	0.0476

From Table 4.4.2, it is clearly shown that the time is decreasing as we increase the number of processors. This is also true for different numbers of Gauss points.

Table 4.4.2: Parallel Execution Time

4 Gauss points					
No Seg	Number of Processors				
	1	3	5	7	9
4	0.004575	0.002627	0.002327	0.002721	0.002859
8	0.012170	0.005897	0.004198	0.005168	0.004759
16	0.039884	0.016981	0.012851	0.011770	0.010601
32	0.207970	0.080007	0.054520	0.043511	0.038995
64	0.571622	0.239573	0.149488	0.119989	0.103278
6 Gauss points					
No Seg	Number of Processors				
	1	3	5	7	9
4	0.006352	0.003254	0.002893	0.003043	0.003293
8	0.017545	0.007934	0.006450	0.006018	0.005832
16	0.057296	0.023327	0.017423	0.015479	0.013409
32	0.207970	0.080007	0.054520	0.043511	0.038995
64	0.811579	0.319298	0.198364	0.157592	0.133485
8 Gauss points					
No Seg	Number of Processors				
	1	3	5	7	9
4	0.007986	0.003991	0.003480	0.003703	0.003949
8	0.022559	0.010016	0.008088	0.007281	0.006645
16	0.074477	0.029384	0.021283	0.019337	0.016902
32	0.270178	0.101775	0.069344	0.053633	0.047436
64	1.048574	0.402817	0.247835	0.195707	0.164163

From Table 4.4.3, we can observe that the speedup is approaching linearity but not growing really close to a linear rate.

Table 4.4.3: Speedup

4 Gauss Points (Linear)				
No of Segments	Number of Processors			
	3	5	7	9
4	1.74	1.97	1.68	1.6
8	2.06	2.47	2.35	2.56
16	2.35	3.10	3.39	3.76
32	2.60	3.81	4.78	5.33
64	2.39	3.82	4.76	5.53
6 Gauss Points (Linear)				
No of Segments	Number of Processors			
	3	5	7	9
4	1.95	2.19	2.09	1.93
8	2.21	2.72	2.92	3.01
16	2.46	3.29	3.70	4.27
32	2.60	3.81	4.78	5.33
64	2.54	4.09	5.15	6.08
8 Gauss Points (Linear)				
No of Segments	Number of Processors			
	3	5	7	9
4	2.00	2.29	2.16	2.02
8	2.25	2.79	3.10	3.39
16	2.53	3.50	3.85	4.41
32	2.65	3.90	5.04	5.70
64	2.60	4.23	5.36	6.39

Table 4.4.4 shows the corresponding efficiency of the algorithm. From the table we can see that the efficiency is approaching 1 as the number of segments is increasing.

Table 4.4.4: Efficiency

4 Gauss Points				
No of Segments	Number of Processors			
	3	5	7	9
4	0.58	0.39	0.24	0.18
8	0.69	0.49	0.34	0.28
16	0.78	0.62	0.48	0.42
32	0.87	0.76	0.68	0.59
64	0.80	0.76	0.68	0.61
6 Gauss Points				
No of Segments	Number of Processors			
	3	5	7	9
4	0.65	0.44	0.30	0.21
8	0.74	0.54	0.42	0.33
16	0.82	0.66	0.53	0.47
32	0.87	0.76	0.68	0.59
64	0.85	0.82	0.74	0.68
8 Gauss Points				
No of Segments	Number of Processors			
	3	5	7	9
4	0.67	0.46	0.31	0.22
8	0.75	0.56	0.44	0.38
16	0.84	0.70	0.55	0.49
32	0.88	0.78	0.72	0.63
64	0.87	0.85	0.77	0.71

Appendix A shows the graphical representation of Table 4.4.1, 4.4.2, 4.4.3 and 4.4.4.

5.0 CONCLUSION

The spherical bubble modules have been applied to the Sequent S30 to study the parallel performance. This study showed that different numbers of processors could significantly affect performance for different numbers of segments and different numbers of Gauss points and the suitability of the algorithm to massive parallelization was shown. From the results produced, we can see parallel computing greatly conserves the computational effort and an effective tool for bubble dynamics problems.

REFERENCES

[1] C. A. Brebbia, *The Boundary Element Method for Engineers*, London, Pentech Press, 1978.

[2] A. J. Davies, "The Boundary Element Method on The ICL DAP", *Parallel Computing*, Vol. 8, 1988, pp. 348-353.

[3] A. J. Davies, *Quadratic Isoparametric Boundary Elements on the ICL DAP - in Boundary Elements X*, ed Brebbia, C.A., Vol. 3, pp. 657-666, Springer-Verlag, 1988.

[4] A. J. Davies, *Mapping the Boundary Element Method to the ICL DAP – in CONPAR 88*, eds Jesshope, C. R. and Reinartz, K. D., 1989, pp. 230-237, Cambridge University Press.

[5] A. J. Davies, *Parallel Computing for the Boundary Element Method - in High-Performance Computing in Engineering. Application to Partial Differential Equation,* eds Power, H. and Brebbia, C. A, Vol. 2, pp. 239-279, Computational Mechanic Publications, 1995.

[6] C. Hasting Jr. *Approximation for Digital Computers*, Princeton University Press, Princeton, N. J, 1955.

[7] M. A. Jaswon and G. T. Symm, *Integral Equation Methods in Potential Theory and Elastostatic*, London, Academic Press, 1977.

[8] J. R. Blake, J. M. Boulton-Stone, and R. P. Tong, *Boundary Integral Methods for Rising, Bursting and Collapsing Bubbles, in BE Applications in Fluid Mechanics*, edited by H. Power,1995, Vol. 4, pp. 31-72.

[9] S. Kim and M. Amann, *Simulation of Micro-structure Evolution on High-Performance Parallel Computer Architectures: Communication Scheduling Strategies for CDL-BIEM-in Boundary Element Technology VII,* eds Brebbia, C. A. and Ingber, M, Elsevier, 1992, pp. 863-872.

[10] P. S. Ramesh,, H-W. Hsu, P. L-F. Liu, and M. H. Lean, *BEM Simulation of Breaking Waves in a Distribute*d *Computing Environment*. Paper presented at IABEM-92, University of Colorado, U. S., 1992.

[11] P. B. Robinson and J. R. Blake, *Dynamics of Cavitation Bubble Interactions* In Bubble Dynamics and Interface Phenomena. Eds. Blake, J. R. and Thomas, N. H. 1994, pp. 55-64.

[12] B. Song, R. Gay and B. Parsons, "Parallel Processing of Quadratic Boundary Elements" *Engng. Anal. with Boundary Elements*, Vol. 11, 1993, pp. 305-311.

[13] G. T. Symm, "Boundary Elements on a Distributed Array Processor", *Engng. Anal*., Vol. 1, 1984, pp. 162-165.

[14] B. Taib, *Boundary Integral Method Applied to Cavitation Bubble Dynamics*, PhD Thesis, University of Wollongong, New South Wales, Australia, 1985.

[15] Y. Cao, W. W. Schultz, and R. F. Beck, "Three Dimensional Desingularized Boundary Integral Method for Potential Problems", *Int. J. Num. Method Fluids*, Vol. 12, 1991, pp. 785-803.

[16] S. G. Akl, *The Design of Parallel Algorithms*, Prentice Hall, 1989.

[17] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing 2nd Edition.* New York, The Benjamin Cummings Pub. Co., Ltd, 1994.

[18] P. K. Banerjee, *The Boundary Element Method Methods in Engineering*, McGraw-Hill Book Company, London, 1994.

[19] C. J. Beckmann and C. D. Polychronopoulus. "The Effect of Scheduling and Synchronization Overhead on Parallel Loop Performance". *CSRD Report No. 1111, Center For Supercomputing Research and Development*, Univ. of Illinois at Urbana-Champaign, 1991.

[20] T. L. Freeman and C. Phillips, *Parallel Numerical Algorithms*, England, Prentice Hall, 1992.

[21] I, Kosztin and K. Schulten, "Boundary Integral Method for Stationary States of Two-Dimensional Quantum Systems". *International Journal of Modern Physics C*, Vol. 8, No. 2, 293-325, 1997.

[22] R. H. Perrott, *Parallel Programming.* Wokingham, Addison Wesley Inc., 1987.

[23] Sequent Computer System, *Sequent Multiprocessor Architecture Overview*. Sequent Computer System, Inc., 1995.

[24] B. Wilkinson and C. M. Allen, *Parallel Programming: Technique and Applications Using Networked Workstations and Parallel Computers.* New Jersey, Prentice Hall, 1999.

[25] R. Abdullah, *Design and Analysis of Numerical Algorithms for the Solution of Linear Systems on Parallel and Distributed Architectures*. PhD Thesis, Loughborough University , United Kingdom, 1997.

[26] M. J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, 1994.

[27] B. P. Lester, *The Art of Parallel Programming*, Prentice Hall, 1993.

APPENDIX A

Fig. A1: Graphical representation of Table 4.4.1

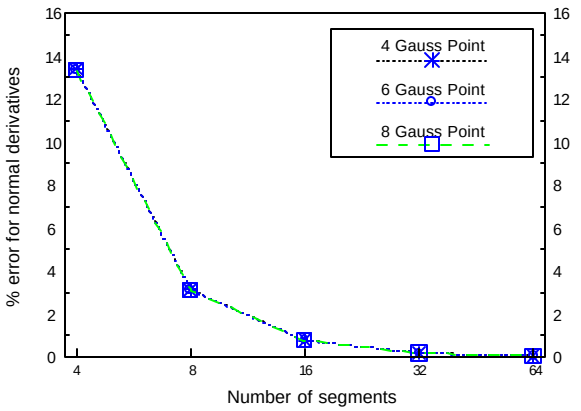
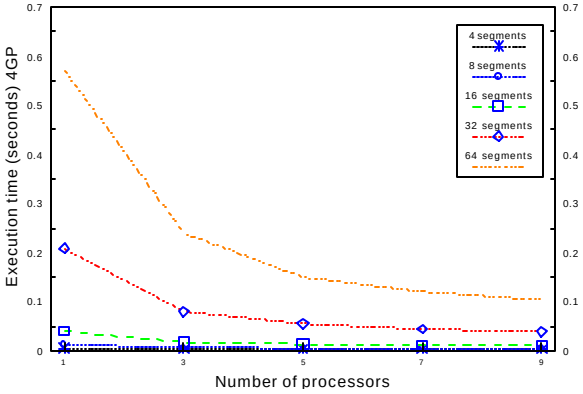
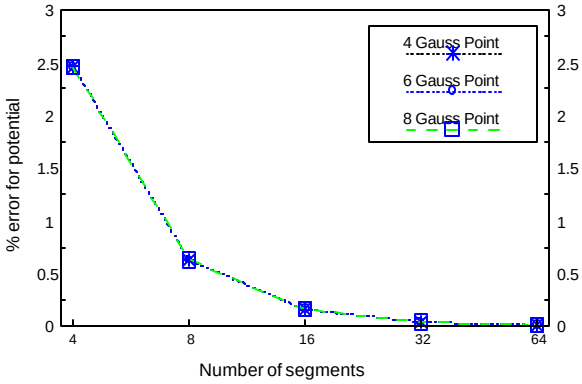
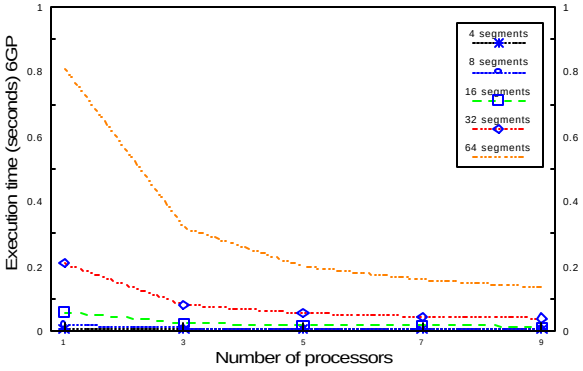
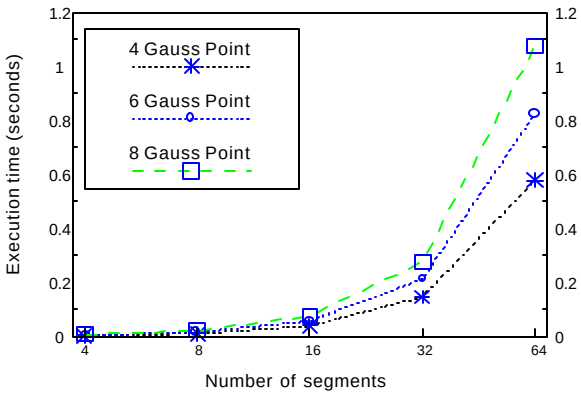


Fig. A2: Graphical representation of Table 4.4.2









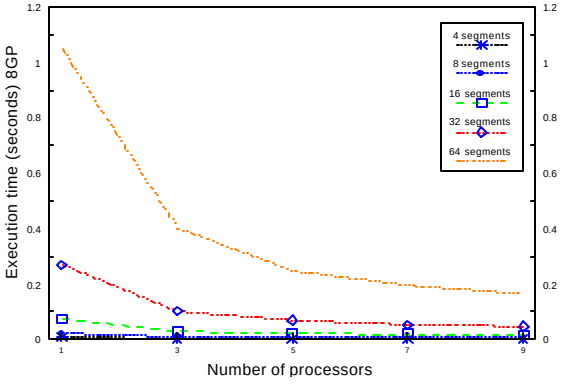
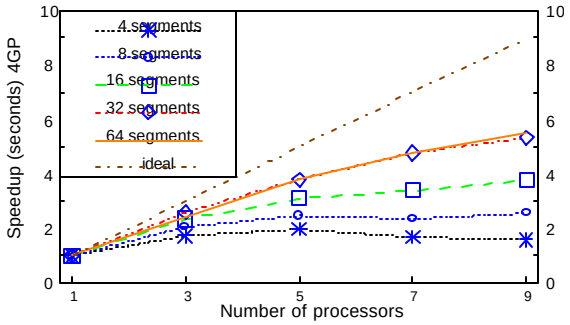
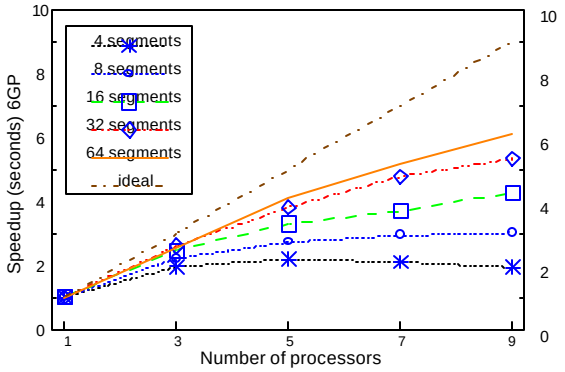


Fig. A3: Graphical representation of Table 4.4.3





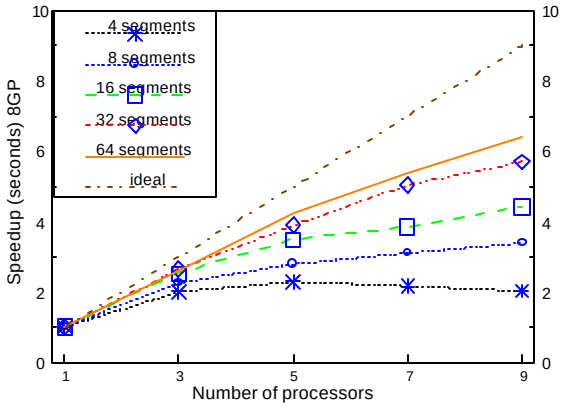
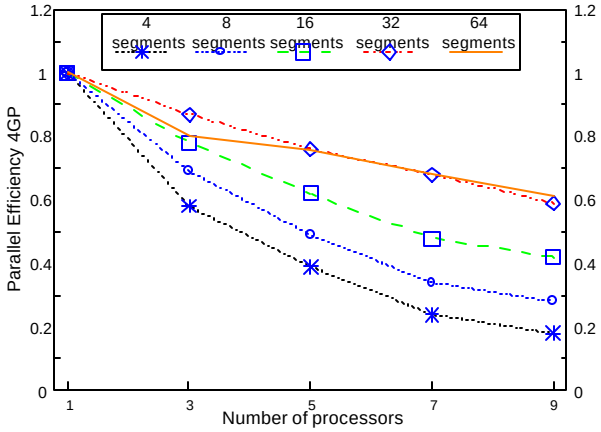
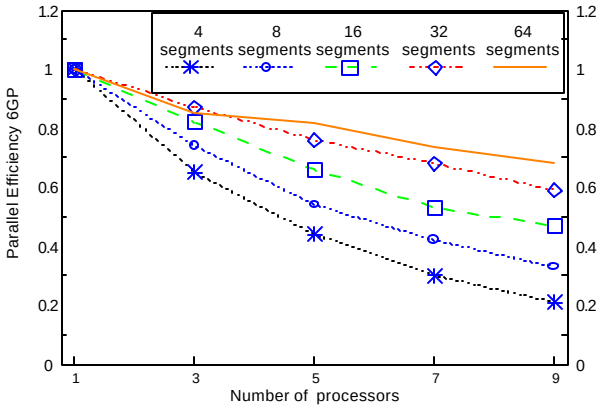
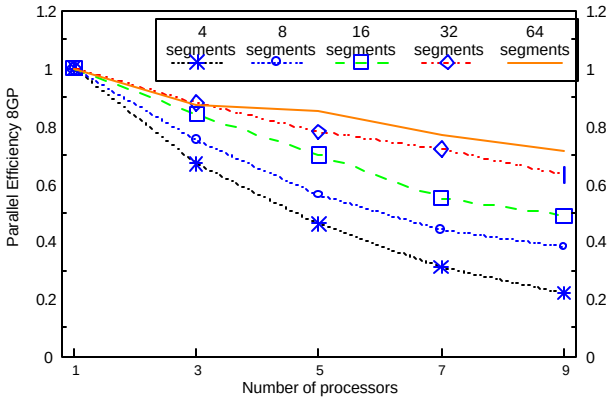


Fig A4: Graphical representation of Table 4.4.4







BIOGRAPHY

Rozita Johari obtained her M.Sc in Computer Science from Illinois Institute of Technology, Chicago in 1987, and currently pursuing her PhD at Universiti Putra Malaysia. Her main research areas include parallel computing and operating system.

Mohd Yazid Mohd Saman obtained his PhD in Parallel Processing from Loughborough University UK in 1993. Research interests include parallel processing, simulation, computer networks and computer graphics.

Bachok M Taib is an Associate Professor and Founding Director of the Software Development Institute, Universiti Putra Malaysia. He holds a PhD degree (1986) in numerical analysis from University of Wollongong, New South Wales, Australia, MSc in numerical analysis (1977) from University of Reading, United Kingdom and B.Sc(Hons)(1974) in mathematics from the National University of Malaysia.

Mohamed Othman completed his PhD from University Kebangsaan Malaysia in 1999. Currently, he is a lecturer at the Department of Communication Technology and Networks, Faculty of Computer Science and Information Technology, University Putra Malaysia. His research interests include parallel computing, high speed network computing (cluster computing), artificial intelligence, expert system design, scientific computing, and programming in logic (parallel).