COMP1680

CLOUDS, GRIDS AND VISUALISATION

NAME: CHARITHA SRI MIKKILI

ID: 001403755

COURSE: MSc Data science

Table of Contents

PART 1

1. Introduction

Multithreading, HPC and Cloud Computing are some of the critical technologies that support computational capabilities for tasks that involve computations and enormous information processing. Multiprocessing is obtained through multi-core and distribution processing, meaning that different tasks can be executed simultaneously, stabilising the computational time for lengthy problems. HPC takes this concept even further with groups of high-performance processors designed for handling large-scale workloads, which may be critical and useful in areas such as scientific experimentation, mathematical modelling, and engineering analysis.

Cloud computing offers a flexible service model for accessing large computation resources through servers at a distance. This reduces the hardware cost and installation, which provides flexibility and opportunity to develop without significant investments at the beginning of the project, which is usually pleasing for SMEs. A small consultancy with 30 consultants, where each of them needs about 1,400 CPU hours per month, is likely to be catered for by an efficient use of either the cloud-based HPC or an onsite HPC put in place. This depends on several variables such as cost, efficiency, expansion, and administration needs Important factors that shall be discussed here are cost efficiency expansion and queen needs.

2. Analysis of Cloud Providers vs. Onsite HPC for Parallel Processing

2.1 Scalability

The use of cloud providers provides a highly scaled solution through service provision, removing the need for additional capital to buy hardware in periods of peak demand to meet the potential increase in consultancy work. This elasticity helps avoid over-provisioning and save money or rather keep costs low for very high and not-so-frequently required resource usage. An onsite HPC is a fixed capability. While it starts with a solid foundation, any further load increase would necessitate additional costly investments in hardware, which can lead to idle capacity at other times.

Accessibility and Flexibility

Cloud platform means access from any location, and thus, the openness promotes work from home, which shall be of great benefit to consultants working at consultancy firms and may be

situated in different places. Cloud services for organisation and productivity targets are designed for seamless operation irrespective of device and without consideration of elaborate configurations. Further, the interfaces they afford for the overall management of resources and tasks are quite flexible. At the same time, onsite HPC systems are usually only available to clients who have access to the resources or need to configure VPN connections, which severely limits their mobility.

Cost Considerations

Cloud computing is a usage-based model, so it is flexible to fluctuating use and overall can avoid the significant capital required to purchase hardware. But if the level of usage remains high and the consultancy estimates 1400 CPU hours per consultant per month, then the costs of the cloud quickly add up. For instance, if AWS Spot Instances cost roughly $0.04 per CPU hour, then each consultant, in the aggregate, would consume $56 monthly for 30 consultants, or roughly $1,680 monthly. On the other hand, an onsite HPC can be expensive at the outset because of the money required upfront to purchase, install and commission the hardware. Because of the negligible usage costs, it might turn out to be more financially justified if the CPU requirements are steady and foreseeable.

Management and Maintenance

The organisational undertakings of infrastructure management and maintenance, upgrade, and support rest on the cloud providers, not the consultancy. AWS, Azure and Google Cloud have ditched the need for patching, handling failed components and scaling. The other disadvantage of an onsite HPC is the necessity of employing IT specialists who shall take care of the maintenance, repair and upgrade of the whole system, which contributes to the increase of operating expenses and number of employees. Of these cost repercussions, the lighter management and overhead costs in the cloud are a boon for a small consultancy company as an investment.

Comparison between Providers

Provider	Key Features	Advantages	Best For
AWS	HPC-focused services such as AWS ParallelCluster and EC2 Spot Instances	Cost-effective for non-urgent batch processing, extensive documentation and compatibility with HPC	Workloads requiring flexibility and broad HPC support
Azure	Batch and HPC services, integrated with Microsoft enterprise tools	Strong integration with Microsoft ecosystem, beneficial for companies using Microsoft software	Companies already using Microsoft tools
Google Cloud	Google Kubernetes Engine (GKE) for containerized workloads, sustained usage discounts	Cost-effective for consistent CPU usage, advantageous for AI and machine learning	AI, machine learning, and containerized tasks

Table 1: Different cloud providers

An example of Cloud costing

A consultant with AWS Spot Instances explained that at $0.04 per CPU hour and 1400 hours of monthly consultant workload, the cost to run is $56. This means if all the 30 consultants are to use the cloud services, then the cost per month would be about $1,680. This makes it possible for the consultancy to scale the costs depending on usage. However, when usage levels are consistently high, the overall costs are equivalent to the outsourcing of one on-site HPC.

3. Cost Comparison: Onsite HPC vs. Cloud

Onsite HPC Costs

Initial Purchase Cost: An onsite HPC cluster to effectively support computations of the 30-consultant team is estimated to cost about $150,000. This estimation comprises fees for high-performance network servers and other related installation components. This investment gives

ownership, eliminating the monthly rental costs, thus useful for consistent, intensive CPU use in the long run.

Recurring Costs: Estimating an annual maintenance and operational cost of $20,000. The energy cost of electricity & cooling of an HPC cluster can even become significantly high. For example, at 300 to $500 per month, this would amount to about $4800 annually. HPC needs to be run by a team of IT professionals. It is estimated that $12,000 of an IT professional's annual salary would go into maintaining and troubleshooting clusters. It is estimated that $3,200 annually for unpredicted hardware refurbishing and components swap would be sufficient to cater for the maintenance requirement.

Depreciation: By dividing the initial outlay of $150,000 by 60 months, which is the prescriptively assumed equipment lifespan in this case, the company suffers a depreciation expense of $2,500 per month. When adding the $20,000 annual maintenance, the overhead cost of an onsite HPC is roughly $4,167 per month.

Total Monthly Cost for Onsite HPC: $4,167

Cloud HPC Costs

Hourly Costs: Consumption-based price structures give cloud providers versatility that allows for low operation costs for fluctuating and temporary workloads. For instance, there are AWS Spot Instances to get a chance to access it at $0.04 per CPU hour. For 30 consultants, each using 1,400 CPU hours monthly, the cost would be:

1,400 hours × 30 consultants × 0.04 USD/hour = 1,680 USD/month

Monthly Variability

On the basis of a particular cloud provider, storage may cost about $ 100-$ 200 per month or more if there is a necessity for the storage of large amounts of data in a particular project. For data-oriented workloads, providers may impose fees for data egress from their cloud, which may incur additional costs depending on the project's requirements.

Total Monthly Cost for Cloud HPC: About $1,680

4. Recommendations and Justification

Platform Recommendation

- Since the cost flexibility and management concerns are concerned, AWS is suggested as being the most optimal of all the four cloud suppliers for this consultancy as the spot pricing option is cheaper and it supports a wide range of HPC.
- If the consultancy appreciates working within Microsoft tools and processes, then Azure would actually be the best option, given that it is fully integrated with the rest of the Microsoft tools.

Rationale

- Cloud computing means that the consultancy is able to rent resources instead of buying hardware, which can be expensive and rigid in terms of offering the organisation the necessary capacity during peak times of work. AWS's EC2 Spot Instances and Azure have the capability for batch processing whenever there is a need for batch workloads.
- Consultants use cloud solutions that grant them access regardless of location and do not require extensive setup at the site of work. Such a feature is highly beneficial for a small consultancy with branches in different locations.
- This means that the consultancy can avoid huge fixed costs associated with physical hosting and instead opt for competitive pricing by AWS, which includes spot instance options. This approach also eliminates deep wallets, and users pay for Internet services proportionately to their utilisations during the month.

Maintenance and Management

By outsourcing the computing solution, onsite maintenance is not required. Therefore, no hPC staffing costs are incurred by the consultancy. AWS and Azure both take care of the infrastructure, upgrades, safety, and maintenance, which is perfect for a group of developers with little to no team focused on IT.

5. Conclusion

AWS is endorsed as the preferred platform for this consultancy owing to cost efficiencies, flexibilities and HPC services. Cloud computing is characterised by sufficient flexibility, accessibility, and low maintenance requirements, which meet the consultancy needs of large, complex processing without the expensive acquisition of specialised infrastructures. In this way, operational needs are fulfilled to a large extent, the current load can be addressed, and further development can be accommodated easily and despite costs.

PART 2

1. Introduction

The aim of this exercise is to tune and apply OpenMP to accelerate a 2D Jacobi iterative numerical solution of the heat conductivity problem on the University's High-Performance Computing (HPC) cluster. The C program "jacobi2d.c" computes heat distribution in a two-dimensional rectangular geometric grid by using the Jacobi method and is adopted mostly in computational mathematics and numerical computations to solve partial differential equations. The exercise involves changing the serial version of the code to implement certain boundary conditions and then parallelising the code using OpenMP for the exploitation of multi-core processors.

The main goals are firstly, to set up the necessary boundary conditions for the heat problem; secondly, to convert the given serial code to OpenMP parallel; and thirdly, to perform performance testing on various problem sizes and number of threads. These steps shall give information as to how much performance gains are possible by parallelising the program.

2. Step 1: Serial Version Modification

2.1 Boundary Conditions Implementation

1. Firstly, the original serial version of the Jacobi iterative heat conductivity code, jacobi2d.c, was adapted to include specific boundary conditions.
2. The chosen working range: the upper constraint at 15°C, the low constraint at 60°C, the left constraint at 47°C and the right at 100°C.
3. Applying these boundary conditions required setting up the initial values of the temperature distribution grid on the edges to these distinct values to maintain their non-change during iterations.
4. As a form of data validation, boundary conditions in C were given fixed values to the edges within the 2D array used to represent the grid.
5. This modification was made by including statements that assign the desired temperature values to the boundary points of the array at the beginning of each iteration, as well as the interior points that are updated by the average value of their neighbours.

6. This step saw to it that the boundary values are retained as the Jacobi iteration determines the heat distribution over the grid before the performance measures are made.

2.2 Execution

```
    }

    int m = atoi(argv[1]);
    int n = atoi(argv[2]);
    double tol = atof(argv[3]);

    double t[m + 2][n + 2], tnew[m + 2][n + 2], diff, difmax;

    // Initialising the temperature array with 11°C everywhere
    for (int i = 0; i <= m + 1; i++) {
        for (int j = 0; j <= n + 1; j++) {
            t[i][j] = 11.0;
        }
    }

    // Setting boundary conditions as per assignment requirements
    for (int i = 1; i <= m; i++) {
        t[i][0] = 47.0;      // Left boundary
        t[i][n + 1] = 100.0; // Right boundary
    }
    for (int j = 1; j <= n; j++) {
        t[0][j] = 15.0;      // Top boundary
        t[m + 1][j] = 60.0;  // Bottom boundary
    }
```

Figure 1: Setting boundaries

Figure 1 defines the function to print a summary of a temperature grid at its boundaries and centre. It then initialises a grid with 11°C everywhere and sets specific boundary conditions. The code implements a numerical method to solve a heat equation problem.

Implements the Jacobi iterative method to solve a 2D steady-state heat conduction problem. It initialises a grid with fixed boundary conditions, iteratively updates the interior points, and calculates the maximum difference between iterations. The process continues until the difference falls below a specified tolerance.

This shows the iteration progress, updating the grid points until convergence, shown by the decreasing maximum difference.

Levels of Optimization	_0	_01	_02	_03
Problem size (100x100)	1.560s	1.550s	1.390s	1.350s
Problem size (150x150)	1.560s	5.440s	5.040s	5.050s
Problem size (225x225)	20.140s	20.830s	20.000s	20.570s
Problem size (275x275)	41.260s	40.790s	40.590s	40.790s

Table: Different iterations and execution times

3. Step 2: Parallelization with OpenMP

3.1 Parallelization Strategy

1. The serial code was parallelised using OpenMP. The parallelisation approach of OpenMP allows computations to be distributed among different threads in cases when they are independent and necessary for large problem sizes.
2. The first modification that was made was that the #pragma omp parallel was added to the loop, which calculates the temperature values across the grid.
3. This directive meant it was possible for multiple threads to be run on different rows where each thread would have the responsibility of working on the rows of the grid.
4. The timers were inserted into the parallelised code for the purpose of making performance comparisons based on the number of threads.
5. From OpenMP's timing functions, such as the omp_get_wtime(), start and end times were taken around the portions of code with parallelisation.
6. This approach allowed for obtaining accurate run time measurements to compare speed up with varied numbers of threads (1, 2, 4 and 8), attesting to the usefulness of parallelism.

3.2 Testing

```
iter;
difmax = 1000000.0;

// Timers for parallel region
double start_time, end_time;
double parallel_runtime;

for (int num_threads = 1; num_threads <= 8; num_threads *= 2) {
    // Set the number of threads
    omp_set_num_threads(num_threads);

     // Initialising the temperature array with 11°C everywhere

    for (i = 0; i <= m + 1; i++) {
        for (j = 0; j <= n + 1; j++) {
            t[i][j] = 11.0;
        }
    }

    // Set boundary conditions
    for (i = 1; i <= m; i++) {
    t[i][0] = 47.0;     // Left boundary
    t[i][n + 1] = 100.0;  // Right boundary
}

    for (j = 1; j <= n; j++) {
    t[0][j] = 15.0;      // Top boundary
    t[m + 1][j] = 60.0;  // Bottom boundary
}
     // Start timing

    start_time = omp_get_wtime();
//Jacobi iteration loop
    while (difmax > tol) {
```

Figure 2: Parallel version using "OpenMP"

Figure 2 parallelises the temperature calculation and array copying loops. The code prints grid summaries and execution time at intervals and after convergence. The parallel execution time is reported at the end.

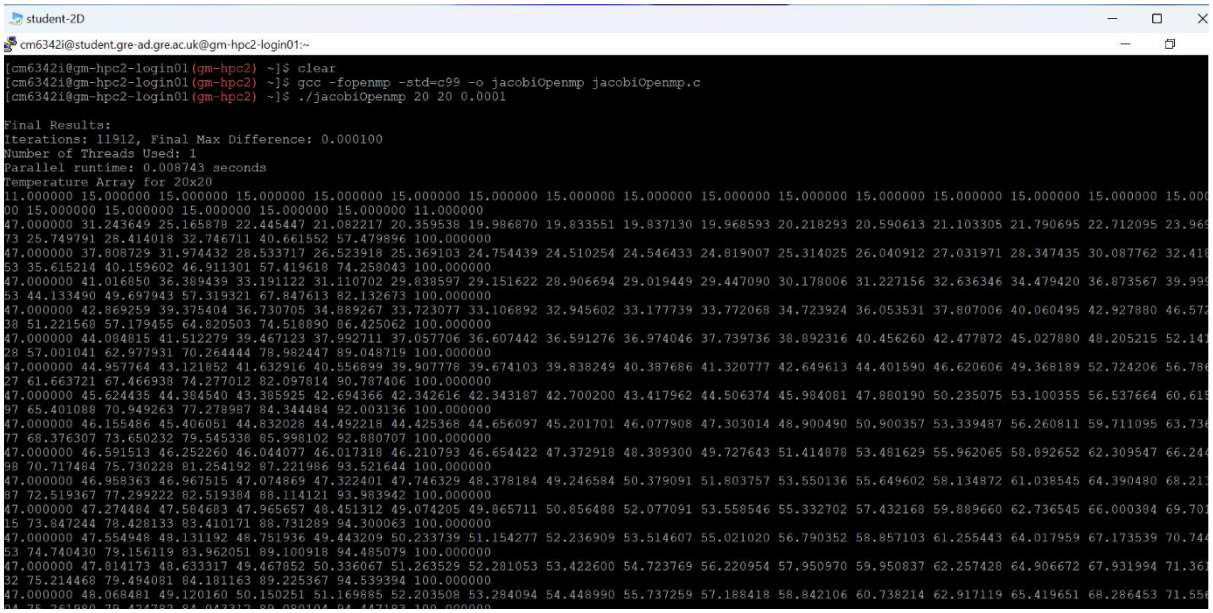

Figure 3: Calculating parallel run-time by using 1 thread

Figure 3 shows solving the 2D steady-state heat conduction problem on a 20x20 grid with a tolerance of 0.1 using OpenMP with one thread. The grid starts with fixed boundary conditions. The iterations progress until convergence, and the final temperature distribution and execution time are reported.

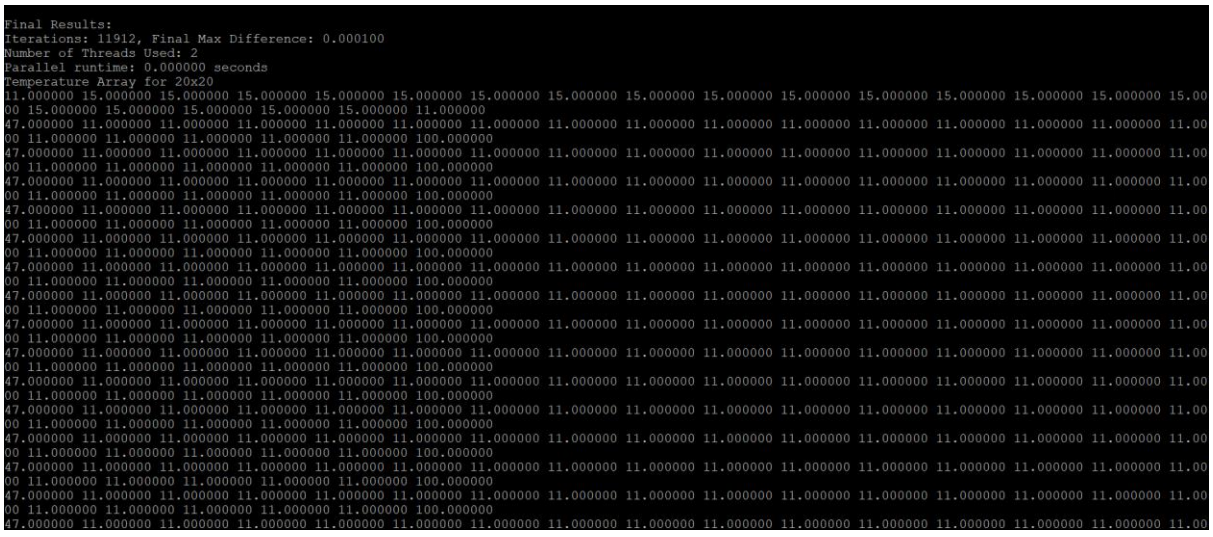

Figure 4: Calculating parallel run-time by using 2 thread

Figure 4 output shows the final temperature distribution and parallel execution time.

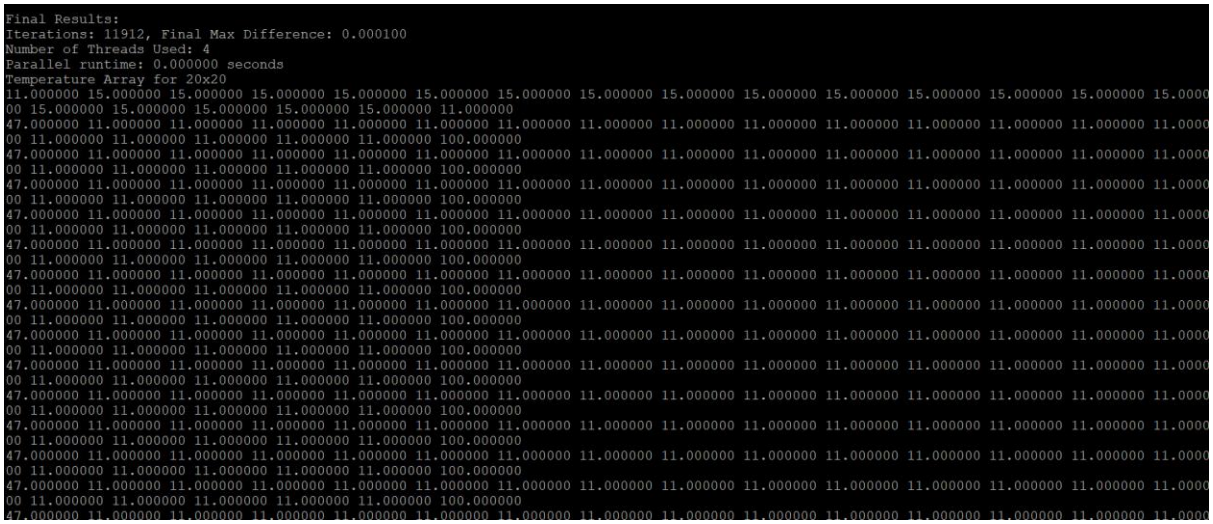

Figure 5: Calculating parallel run-time by using 4 thread

Figure 5 shows the output wherein the grid starts with fixed boundary conditions. The iterations progress until convergence, and the final temperature distribution and parallel execution time are reported.

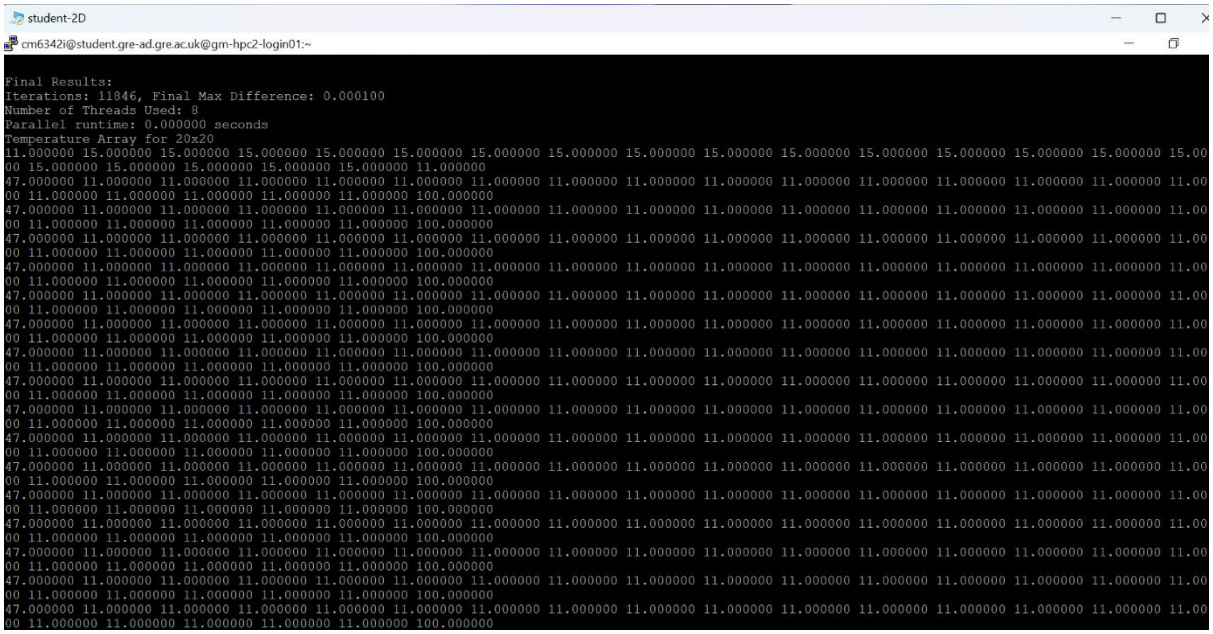

Figure 6: Calculating parallel run-time by using 8-thread

Figure 6 output shows the Jacobi iterative method. As the number of threads increases, the parallel execution time decreases.

4. Step 3: Performance Testing on HPC

4.1 HPC Execution and SLURM Queue

1. The OpenMP version of the code was compiled and run on the University's HPC using the SLURM job scheduler.
2. SLURM makes job management easier through resource allocation as well as dealing with multiple users. Thus, it is effective for the running of tests across various threads.
3. SLURM was employed to write batch scripts that involved parameters, including a number of threads (2, 4, and 8) and the available system resources.
4. To enhance the run time efficiency and to get nearer to real-time complexity, comment statements used in the programs were minimal because such comments could slow down the runtime vastly.
5. The problem size was scaled up to guarantee that the workload would capture speedup improvements on the distributed system.
6. Due to the larger problem size and restricted printout, the evaluation of the scalability and speed of the code on HPC is much more effective.
7. Metrics were gathered and calculated during runtime regarding how parallelism was implemented and the resulting performance when threading and workload size were varied in the HPC setting.

```
            // Update the global difmax after the reduction
            #pragma omp critical
            if (local_difmax > difmax) {
                difmax = local_difmax;
            }

            // Copy new to old temperatures
            #pragma omp parallel for shared(t, tnew) private(i, j)
            for (int i = 1; i <= m; i++) {
                for (int j = 1; j <= n; j++) {
                    t[i][j] = tnew[i][j];
                }
            }
        }

        end_time = omp_get_wtime();
        parallel_runtime = end_time - start_time;

        // Print results
       printf("Iterations: %d, Final Max Difference: %lf\n", iter, difmax);
       printf("Number of Threads Used: %d\n", num_threads);
       printf("Parallel runtime: %lf seconds\n", parallel_runtime);


        // Calculate speedup and theoretical maximum speedup
        double speedup = serial_runtime / parallel_runtime;
        printf("Speedup: %lf\n", speedup);

        double theoretical_max_speedup = 1.0 / ((1.0 - 1.0 / num_threads) + 1.0 / num_threads);
        printf("Theoretical Maximum Speedup: %lf\n", theoretical_max_speedup);

        for (i = 0; i <= m + 1; i++) {
            for (j = 0; j <= n + 1; j++) {
              printf("%3.3lf ", t[i][j]);
            }
            printf("\n");
        }

        // Reset iteration count and maximum difference for the next test
        iter = 0;
        difmax = 1000000.0;
    }

    return 0;
}

void runOptimizedSerial(int m, int n, double tol) {
    // Implementation of the optimized serial version
```

Figure 7: Setting HPC and the SLURM queue

Figure 7 code prints the number of iterations, maximum difference, and parallel execution time after convergence. It also includes memory management for dynamically allocated arrays.

4.2 Speedup Results

uses varying numbers of threads (1, 2, 4, and 8). The speedup of the parallel execution time is observed with an increasing number of threads.

Levels of optimization	**Problem sizes**	**Execution time[step-1]**	**Execution time[step-3]**	**Speed up [thread 2]**	**Speed up [thread 4]**	**Speed up [thread 8]**
_o	100	1.560s	Thread 2=0.908s Thread 4=0.605s Thread 8=0.332s	1.718s	2.578s	4.698s
_o	150	5.170.s	Thread 2=3.127s Thread 4=1.776s Thread 8=0.973s	1.653s	2.911s	5.313s
_o	225	20.140s	Thread 2=11.935s Thread 4=6.607s Thread 8=3.993s	1.687s	3.048s	5.043s
_o	275	41.260s	Thread 2=21.317s Thread 4=11.563s Thread 8=6.080s	1.935s	3.568s	6.786s
_o1	100	1.550s	Thread 2=0.859s Thread 4=0.572s Thread 8=0.269s	1.804s	2.709s	5.762s
_o1	150	5.440s	Thread 2=3.108s Thread 4=1.767s Thread 8=0.964s	1.750s	3.078s	5.643s
_o1	225	20.830s	Thread 2=12.127s Thread 4=6.020s Thread 8=4.055s	1.717s	3.460s	5.136s
_o1	250	40.790s	Thread 2=20.867s Thread 4=11.217s Thread 8=5.847s	1.954s	3.636s	6.976s
_o2	100	1.390s	Thread 2=0.875s	1.588s	2.344s	4.744s

			Thread 4=0.593s Thread 8=0.293s			
_o2	150	5.040s	Thread 2=3.219s Thread 4=1.784s Thread 8=0.971s	1.565s	2.825s	5.190s
_o2	225	20.570s	Thread 2=11.707s Thread 4=6.383s Thread 8=3.674s	1.757s	3.222s	5.598s
_o2	275	40.590s	Thread 2=23.817s Thread 4=12.494s Thread 8=6.875s	1.704s	3.248s	5.904S
_o3	100	1.350s	Thread 2=0.880s Thread 4=0.449s Thread 8=0.265s	1.534s	3.006s	5.094s
_o3	150	5.050s	Thread 2=3.002s Thread 4=1.787s Thread 8=0.998s	1.6822s	2.825s	5.060s
_o3	225	20.570s	Thread 2=10.949s Thread 4=5.520s Thread 8=3.845s	1.878s	3.726s	5.349s
_o3	275	40.790s	Thread 2=23.026s Thread 4=13.328s Thread 8=6.322s	1.771s	3.060s	6.452s

The report shows how parallel programming makes computation faster for intensive numerical simulations such as the Jacobi iterative heat conductivity method. OpenMP integration and the application on the HPC offered a perspective into how much time could be saved through the use of multithreading. The gain of speedup with the increased thread count supports the effect of the parallelism of computations, especially for large sizes of problems. Some issues include managing boundary conditions and iteration stability relating to synchronisation and validation, respectively. This process shows that in parallel computing, time-consuming algorithmic methods might be efficiently solved, especially in high-performance systems.

Also, using software print statements at the optimised level & consistent care for array access in the full system was another sequence that saved some performance points. When testing on multiple thread configurations, this provided the answer to how to scale performance rate increases with progress in thread quantity. This is essential when allocating computing resources. In general, this process demonstrates the ability of parallel computing to improve time-extensive algorithms, especially in a high-performance computing environment where time and scalability are factors. The necessity of parallel computing can be explained by the fact that it is due to its efficiency for powerful methods of managing complex simulations in scientific and engineering computations.

6. Conclusion

Part B examined the enhancement and paralleling of a 2D Jacobi iterative heat conductivity model to show that enhancement of the code is possible using alteration and the OpenMP library. Establishing boundary conditions, dividing the computation and analysing the performance on the University HPC provided proof that parallelism pays off in terms of the reduced runtime, especially for larger grids. Besides the improvement of computational speed and the demonstration of real-life usage of parallel computing in solving demanding simulation problems, the key features of multithreading and optimisation were successfully implemented and applied. In this approach, the usefulness of HPC resources and the use of parallel programming in scientific computing is emphasised.

Bibliography:

Ageed, Z., Mahmood, M.R., Sadeeq, M., Abdulrazzaq, M.B. and Dino, H., 2020. Cloud computing resources impacts on heavy-load parallel processing approaches. *IOSR Journal of Computer Engineering (IOSR-JCE)*, *22*(3), pp.30-41. https://www.researchgate.net/profile/Zainab-Ageed/publication/342379111_Cloud_Computing_Resources_Impacts_on_Heavy-Load_Parallel_Processing_Approaches/links/5ef89156299bf18816eded3a/Cloud-Computing-Resources-Impacts-on-Heavy-Load-Parallel-Processing-Approaches.pdf

Ahmad, A.S., Masa-Ibi, E. and Aliyu, A., Solutions by Optimization of the 2-Dimensional Heat Conductivity Problem on Grid Machines using C++ and OpenMP. *International Journal of Computer Applications*, *975*, p.8887. https://www.academia.edu/download/95946809/ijca2020920007.pdf

Aldinucci, M., Cesare, V., Colonnelli, I., Martinelli, A.R., Mittone, G., Cantalupo, B., Cavazzoni, C. and Drocco, M., 2021. Practical parallelization of scientific applications with OpenMP, OpenACC and MPI. *Journal of parallel and distributed computing*, *157*, pp.13-29. https://www.sciencedirect.com/science/article/pii/S0743731521001295

Ali, S., Wadho, S.A., Yichiet, A., Gan, M.L. and Lee, C.K., 2024. Advancing cloud security: Unveiling the protective potential of homomorphic secret sharing in secure cloud computing. *Egyptian Informatics Journal*, *27*, p.100519. https://www.sciencedirect.com/science/article/pii/S1110866524000823

Cérin, C., Grenèche, N. and Menouer, T., 2023. Executing Traditional HPC Application Code in Cloud with Containerized Job Schedulers. In *High Performance Computing in Clouds: Moving HPC Applications to a Scalable and Cost-Effective Environment* (pp. 75-97). Cham: Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-031-29769-4_5

Chadha, M., John, J. and Gerndt, M., 2020, December. Extending slurm for dynamic resource-aware adaptive batch scheduling. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)* (pp. 223-232). IEEE. https://ieeexplore.ieee.org/abstract/document/9406729/

Demchenko, Y., Cuadrado-Gallego, J.J., Chertov, O. and Aleksandrova, M., 2024. Cloud and Big Data Service Providers and Platforms. In *Big Data Infrastructure Technologies for Data Analytics: Scaling Data Science Applications for Continuous Growth* (pp. 115-144). Cham: Springer Nature Switzerland. https://link.springer.com/chapter/10.1007/978-3-031-69366-3_4

Gypser, P., 2022. *Mechanisms for energy-efficient processor allocation and redistribution on manycore systems* (Doctoral dissertation, BTU Cottbus-Senftenberg). https://opus4.kobv.de/opus4-btu/frontdoor/index/index/docId/6385

Harris, A., 2022. *Qualitative Case Study of Remote Small Business Needs Assessment and Business Development Process Improvement for a Work from Home Writing Consulting Company* (Doctoral dissertation, Trident University International). https://search.proquest.com/openview/d3ad4b96ffb2cc02bbac68b73e9ac5c1/1?pq-origsite=gscholar&cbl=18750&diss=y

Iwasaki, S., Amer, A., Taura, K., Seo, S. and Balaji, P., 2019, September. BOLT: Optimizing OpenMP parallel regions with user-level threads. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*(pp. 29-42). IEEE. https://ieeexplore.ieee.org/abstract/document/8891628/

Lofstead, G.F. and Duplyakin, D., 2021. *Take Me to the Clouds Above: Bridging On Site HPC with Clouds for Capacity Workloads* (No. SAND2021-3399C). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States). https://www.osti.gov/servlets/purl/1856762

Maray, M. and Shuja, J., 2022. [Retracted] Computation Offloading in Mobile Cloud Computing and Mobile Edge Computing: Survey, Taxonomy, and Open Issues. *Mobile Information Systems, 2022*(1), p.1121822. https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1121822

McHaney, R., 2021. Cloud technologies: an overview of cloud computing technologies for Managers. https://books.google.com/books?hl=en&lr=&id=m9QnEAAAQBAJ&oi=fnd&pg=PR13&dq=Cloud+computing+means+that+the+consultancy+is+able+to+rent+resources+instead+of+buying+hardware+which+can+be+expensive+and+rigid+in+terms+of+offering+the+organisat

ion+the+necessary+capacity+during+peak+time+of+work&ots=MaM0aW8EiA&sig=dXDeiVzZmq5QO6ZsKLSI3kyJjRs

Peiris, C., Pillai, B. and Kudrati, A., 2021. *Threat Hunting in the Cloud: Defending AWS, Azure and Other Cloud Platforms Against Cyberattacks*. John Wiley & Sons. https://books.google.com/books?hl=en&lr=&id=RHpAEAAAQBAJ&oi=fnd&pg=PP19&dq=The+organisational+undertakings+of+infrastructure+management+and+maintenance,+upgrade,+and+support+rest+on+the+cloud+providers,+not+the+consultancy.+AWS,+Azure+and+Google+Cloud+have+ditched+the+need+for+patching,+handling+of+failed+components+and+scaling.&ots=7YmNw7ScNX&sig=14fE3GfhP6ZjYfTNx5EZYi-eG30

Pereira, R., 2023. *Efficient Use of Task-based Parallelism in HPC Parallel Applications* (Doctoral dissertation, Ecole normale supérieure de lyon-ENS LYON). https://theses.hal.science/tel-04466797/

Simakov, N.A., Innus, M.D., Jones, M.D., DeLeon, R.L., White, J.P., Gallo, S.M., Patra, A.K. and Furlani, T.R., 2018. A slurm simulator: Implementation and parametric analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017, Proceedings 8* (pp. 197-217). Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-319-72971-8_10

Singh, A.K. and Singh, K.M., 2024. OpenMP-based parallel MLPG solver for analysis of heat conduction. *Engineering Computations*, *41*(2), pp.364-384. https://www.emerald.com/insight/content/doi/10.1108/EC-01-2023-0012/full/html

Staevsky, N. and Gaftandzhieva, S., 2023. Cloud Migration: Identifying the Sources of Potential Technical Challenges and Issues. *International Journal of Advanced Computer Science & Applications*, *14*(12). https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=2158107X&AN=174588375&h=7XUO96c5tqfsx0yMp7BjGaoC7z4Ri82Py2Hgo9b7AD3aQlL%2FVFkSuYIGhq3h%2FlPUwrX3TPVuEI2B1xl%2FImn6Kw%3D%3D&crl=c

Stam, J.I., The factors determining cloud platform adoption. https://repository.tudelft.nl/file/File_d6445a67-9f30-4af6-a42b-0f79cf253f60

Sterling, T., Anderson, M. and Brodowicz, M., 2017. A survey: runtime software systems for high performance computing. *Supercomputing Frontiers and Innovations*, *4*(1), pp.48-68. https://www.superfri.org/index.php/superfri/article/view/126

Zhang, S., Pandey, A., Luo, X., Powell, M., Banerji, R., Fan, L., Parchure, A. and Luzcando, E., 2022. Practical adoption of cloud computing in power systems—Drivers, challenges, guidance, and real-world use cases. *IEEE Transactions on Smart Grid*, *13*(3), pp.2390-2411. https://ieeexplore.ieee.org/abstract/document/9703493/