DHRUV GARG 16 BCE 1190

HIGH PERFORMANCE COMPUTING

DIGITAL ASSIGNMENT 2

QUESTION 1

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. Write a parallel algorithm (not a program) for the above problem and analyse the same.

SOLUTION

Underlying concept: **BACKTRACKING** algorithm

METHOD 1: Iterative method

a) Place the queen at some arbitrary position
b) Check if there exists a queen that could attack this new queen (left, right, diagonally).
c) If no queen will attack: Queen can be placed here. Increment the global counter.
d) If a queen will attack: Backtrack - go back and change the position of the previous queen
e) Stop and print the matrix once all the queens have been placed ie. counter equals N.

METHOD 2: Parallel method

Implementation 1: Using Open-MP (Shared memory architecture)

```
Algorithm
        n_queens
Input
        a number N
Output
        Matrix N x N with position of the N queens

Start
        Initialize I = 0, J = 0
        Use parallel constructs like "sections" or "parallel for" to run for all the queens.
        Run a for loop from I = 0 to I = N (number of queens)
                Each thread can be parallelized into multiple threads
                Assign task to sub threads.
                Use parallel constructs like "sections" or "parallel for"
                Run a for loop from J = 0 to J = N
                        if(board[I][J] is safe and not occupied another queen)
                                Mark the location
                                make board[I][J]=1
                        else
                                Check for (N-1)th queen
                End inner for loop
        End outer for loop
        Print the chessboard matrix
```

Implementation 2: Using MPI (distributed memory architecture)

Algorithm

```
n_queens
```

Input

```
a number N
```

Output

```
Matrix N x N with position of the N queens
```

```
Start
    Initialize I = 0, J = 0
    A master process  can assign the work to the slave threads
    Run a for loop from I = 0 to I = N (number of queens)
        Each thread assigned work can be a master for other threads it will create
        Assign work to slave threads under it
        The loop here can use point to point communication or broadcasting
        Run a for loop from J = 0 to J = N
            if(board[I][J] is safe and not occupied another queen)
                Mark the location
                make board[I][J]=1
            else
                Check for (N-1)th queen
        End inner for loop
    End outer for loop
    Print the chessboard matrix
```

QUESTION 2

What are the most important performance metrics for HPC applications?

SOLUTION

A. BENCHMARKS FOR PERFORMANCE OF HPC SYSTEMS

The performance of complex applications on HPC systems can depend on a variety of independent performance attributes of the hardware. The **HPC Challenge Benchmark** is an effort to improve visibility into this multidimensional space by combining the measurement of several of these attributes into a single program.
Although the performance attributes of interest are not specific to any particular computer architecture, the reference implementation of the HPC Challenge Benchmark in **C and MPI** assumes that the system under test is a cluster of shared memory multiprocessor systems connected by a network. Due to this assumption of a hierarchical system structure most of the tests are run in several different modes of operation.

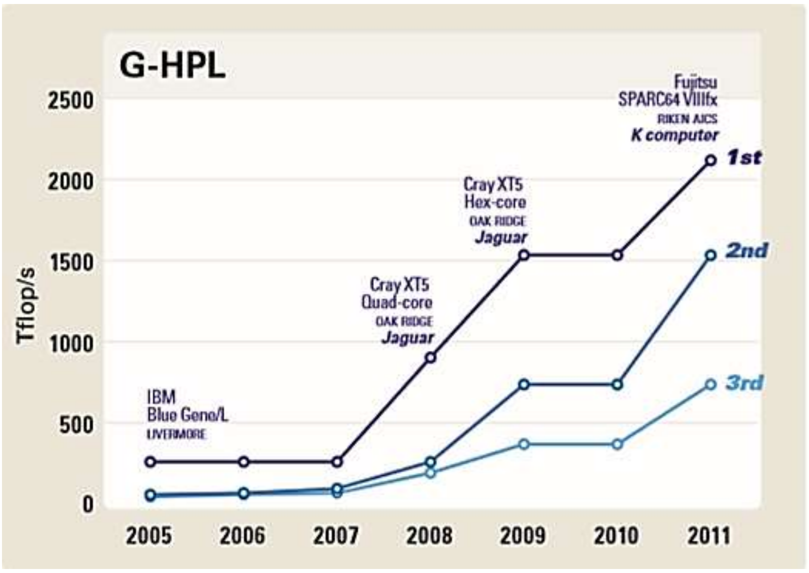
The HPC Challenge benchmark consists of **7 tests**:

DHRUV GARG 16 BCE 1190

1. HPL (High Performance Linpack)

HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark.

The HPL package provides a testing and timing program to **quantify the accuracy** of the obtained solution as well as the time it took to compute it. The best **performance** achievable by this software on your system depends on a large variety of factors. Nonetheless, with some restrictive assumptions on the interconnection network, the algorithm described here and its attached **implementation are scalable** in the sense that their parallel efficiency is maintained constant with respect to the per processor memory usage.



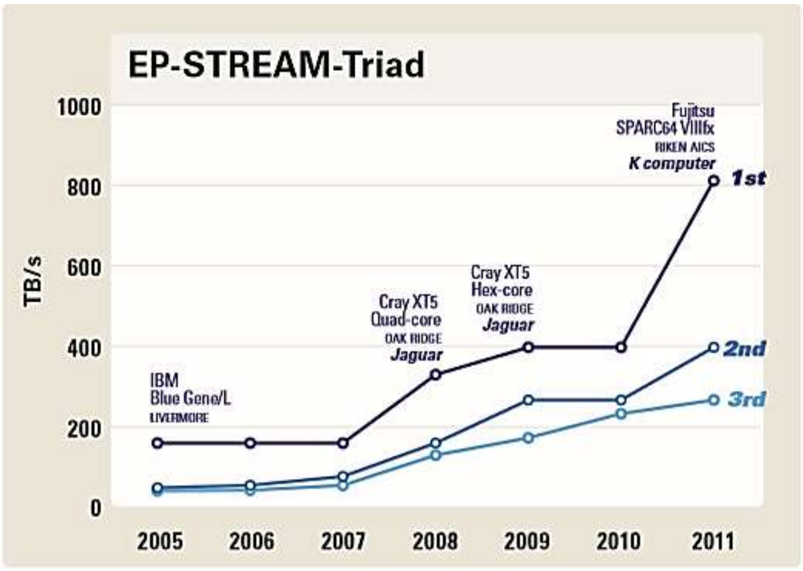
2. STREAM

A simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel. It measures sustained memory bandwidth to/from memory (single, star). STREAM performs 4 simple kernels:

COPY:	$c \leftarrow$	a
SCALE:	$b \leftarrow$	$\alpha\, c$
ADD:	$c \leftarrow$	$a + b$
TRIAD:	$a \leftarrow$	$b + \alpha\, c$

The STREAM run rules require that the data dependency chain implied by the sequence of operations be maintained:

a. Copy
b. Scale
c. Add
d. Triad



3. **PTRANS (parallel matrix transpose)**

It exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network and system interconnect.

The performed operation sets a random n by n matrix to a sum of its transpose with another random matrix:

$$\boxed{A \leftarrow A^T + B}$$

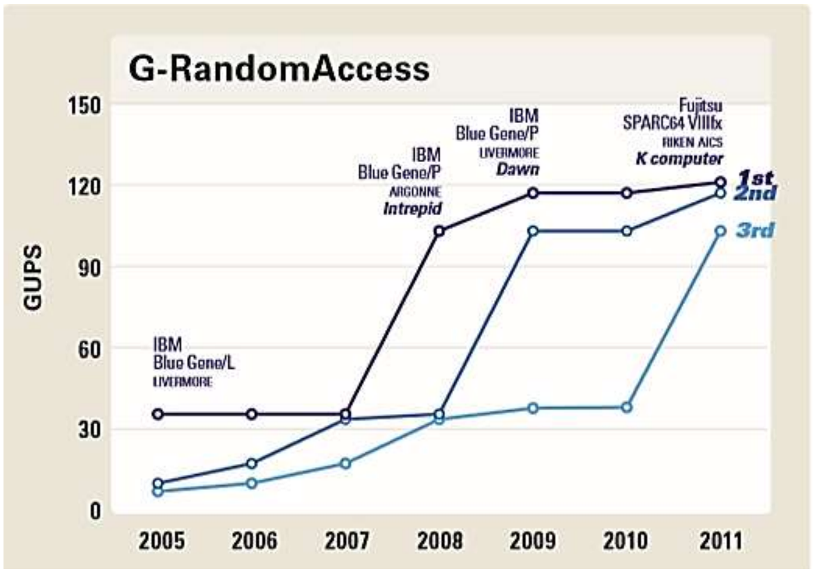
4. **RandomAccess**

RandomAccess measures the rate of integer updates to random memory locations measured by the metric GigaUpdates per Second (GUPS).

The verification procedure allows 1% of the operations to be incorrect (skipped or due to data race conditions) which allows loosening concurrent memory update semantics on shared memory architectures.

The operation being performed on an integer array of size m is:

$$x \leftarrow f(x)$$
$$f : x \mapsto (x \oplus a_i); \quad a_i \text{– pseudo-random sequence}$$

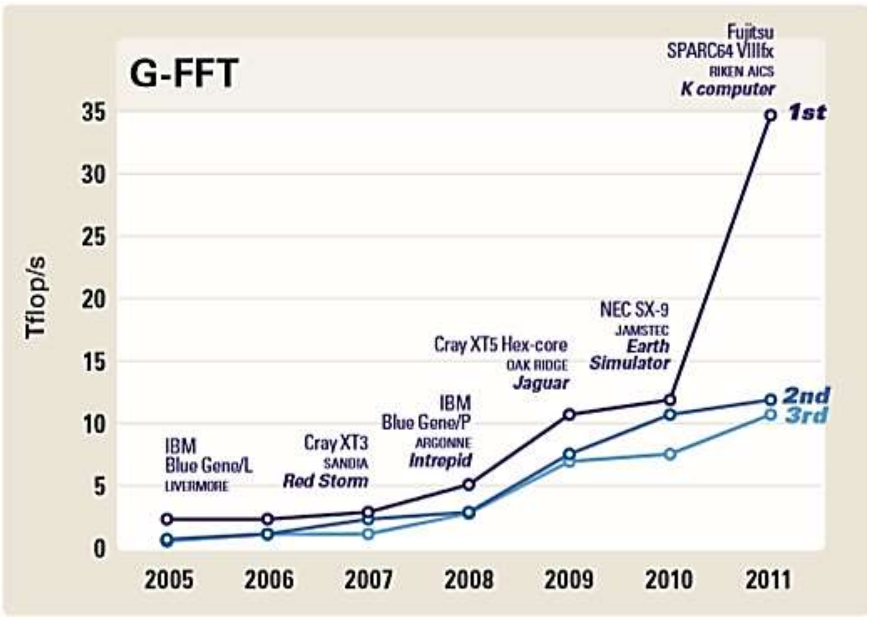


5. **FFT**

FFT measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT) of size m measured in Gflop/s.

The DFT is given by:

$$\boxed{Z_k \leftarrow \sum_{j}^{m} z_j e^{-2\pi i \frac{jk}{m}}; \quad 1 \leq k \leq m}$$



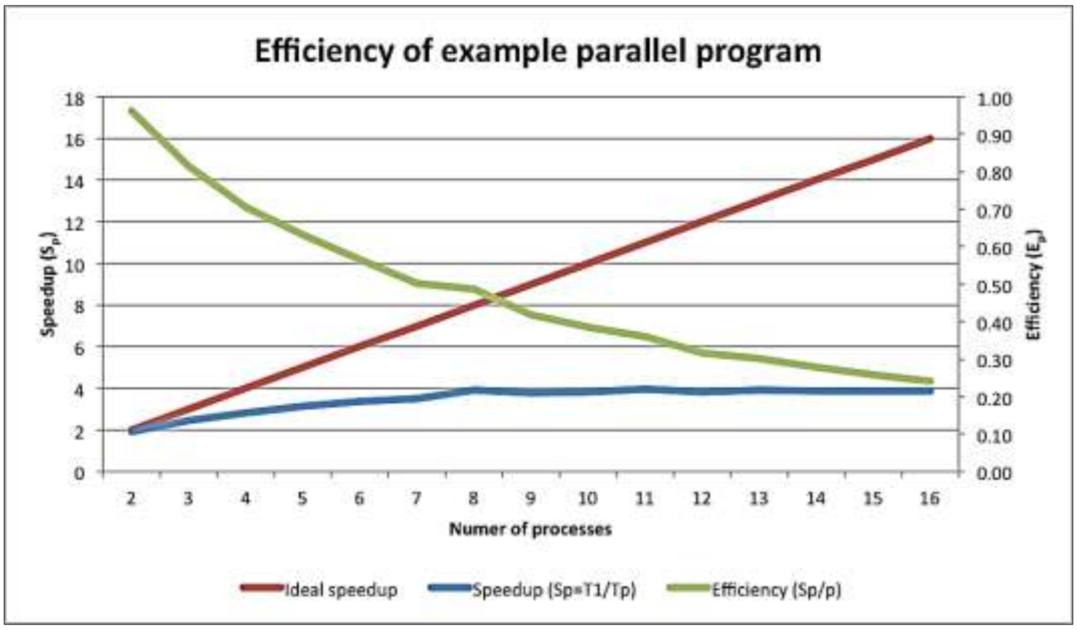
6. **DGEMM**
   Measures the floating point rate of execution of double precision real matrix-matrix multiplication.

7. **Communication bandwidth and latency**
   A set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff (effective bandwidth benchmark).

B. PERFORMANCE METRICS FOR PARALLEL APPLICATIONS IN HPC

There are a number of metrics, the best known are:

1. **Speedup**
   It measures the ration between the sequential execution time and the parallel execution time.
2. **Efficiency**
   Efficiency is a measure of the usage of the computational resources. It measures the ration between performance and the resources used to achieve that performance.



3. **Redundancy**
   Redundancy measures the increase in the required computation when using more processors. It measures the ration between the number of operations performed by the parallel execution and by the sequential execution.
4. **Utilization**
   It is measure of the good use of the computational capacity. It measures the ratio between the computational capacity utilized during execution and the capacity that was available.
5. **Quality**
   Quality is a measure of the relevancy of using parallel computing.

SOME LAWS FOR PERFORMANCE OF PARALLEL ALGORITHMS

There also **some laws/metrics** that tries to explain and assert the **potential performance of a parallel application.** The best known are:

a. **Amdahl's law** can be formulated in the following way:

$$S_{\text{latency}}(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

- S_{latency} is the theoretical speedup of the execution of the whole task;
- *s* is the speedup of the part of the task that benefits from improved system resources;
- *p* is the proportion of execution time that the part benefiting from improved resources originally occupied.

b. **Gustafson Law**

Gustafson estimated the speedup *S* gained by using *N* processors (instead of just one) for a task with a serial fraction *s* (which does not benefit from parallelism) as follows:

$$S = N + (1-N)s$$

Using different variables, Gustafson's law can be formulated the following way:

$$S_{\text{latency}}(s) = 1 - p + sp,$$

- S_{latency} is the theoretical speedup in latency of the execution of the whole task;
- *s* is the speedup in latency of the execution of the part of the task that benefits from the improvement of the resources of the system;
- *p* is the percentage of the execution workload of the whole task concerning the part that benefits from the improvement of the resources of the system *before the improvement*.

Karp–Flatt metric

Given a parallel computation exhibiting **speedup** and **p** processors, where **p** > 1, the experimentally determined serial fraction **e** is defined to be the Karp–Flatt Metric. The lesser the value of **e** the better the parallelization.

$$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Isoefficiency Metric

1. For a given problem size, as we increase the number of processing elements, the overall efficiency of the parallel system goes down. This phenomenon is common to all parallel systems.
2. In many cases, the efficiency of a parallel system increases if the problem size is increased while keeping the number of processing elements constant.

Parallel execution time can be expressed as a function of problem size, overhead function, and the number of processing elements. Equations in Isoefficiency metric:

$$T_P = \frac{W + T_o(W, p)}{p}$$

$$\begin{aligned} S &= \frac{W}{T_P} \\ &= \frac{Wp}{W + T_o(W, p)}. \end{aligned}$$

$$\begin{aligned} E &= \frac{S}{p} \\ &= \frac{W}{W + T_o(W, p)} \\ &= \frac{1}{1 + T_o(W, p)/W}. \end{aligned}$$

QUESTION 3

What steps could be taken to build wider collaboration among HPC sites?

SOLUTION

1. Explicit funding for centre collaboration
   a. Discussing the need with program managers / funding agencies, will also help community get resources to develop these tools.
   b. There are mostly small sites with limited staff which leaves time for being involved in the collaborations liked.
2. Collaboration on building software & software stacks
   a. A robust and standardized installation procedure, which removes "customization" for path, environment vars etc from documentations and others.
   b. Contributing recipes / configuration management playbooks into a central repository.
   c. The OpenHPC initiative's repository could perhaps provide a good platform for this.
3. Community, evangelism, mailing lists
   a. Just evangelism
   b. Community involvement
   c. Mailing lists targeted at support staff
   d. A common mailing list specific to HPC user support
4. Open source tools
   a. Open source tools with good documentation
   b. Ensuring that tools are flexible so that the can be adapted to the eccentricities at each site is also important.
   c. Normalize support tool stack
5. Use and contribute to others' tools
   a. It would help if sites took a look at existing stuff before creating new tools. Many performance monitoring tools are already out there (collectl, collectd, ganglia, etc.)
   b. Sites using open tools, and not re-writing tools every time. For example: taccstats, collectd, ganglia, nwperf, etc.