Performance Results of The NAS Parallel Benchmarks in SISAL

Hung-Yu Tseng and Jean-Luc Gaudiot

CENG Technical Report 93-42

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4484

October 1993

Abstract

This report presents the SISAL performance results of a set of benchmarks developed by NASA at Ames Research Center. These benchmarks are derived from large scale computational fluid dynamics applications for performance evaluation of parallel computers. The computers utilized for this experiment are SUN-4, CRAY C90, CRAY YMP, and Sequent Symmetry.

1 Introduction

Competitive performance of SISAL codes with respect to that of FORTRAN codes are reported here. However,it should be noted that most applications used in the experiments are of small sizes. Even though a data structure can be efficient for one isolated module, it may not be so for larger scientific applications such as the CFD applications. Therefore, experiments are still required with more realistic applications.

The NAS parallel benchmarks consist of a set of kernels and three simulated CFD applications. We implemented the benchmarks in SISAL language. In our SISAL implementation, we used the algorithms employed in the FORTRAN codes proposed by the NASA Ames research center in order to compare the performance between the two languages. We conducted the performance evaluation of SISAL using applications with larger datasets, along with an analysis and development of algorithms and data stuctures particularly suitable for the functional languages.

2 List of benchmarks

The NAS parallel benchmarks[1] include

2.1 The Embarrassingly Parallel Benchmark(EP)

In this benchmark , two dimentional statistics are accumulated from a large number of Gaussian pseudoramdom numbers. It tests the limits for the floating-point performance without much interprocessor communication.

Algorithm:

Produce the pseudorandom floating point r_j in the interaval $(0, 1)$
$a = 5^{13}, r_0 = 271828183, r_{k+1} = a \cdot r_k (mod\, 2^{46})$, for $1 \leq j \leq 2n$.
$k = 0$
for j = 1 to n
$x_j = 2r_{2j-1} - 1$
$y_j = 2r_{2j} - 1$
$t_j = x_j^2 + y_j^2$
if $t_j \leq 1$

$k = k + 1$

$X_k = x_j\sqrt{(-2\log t_j)/t_j}$

$Y_k = y_j\sqrt{(-2\log t_j)/t_j}$

end if

end for

Set $Q_l = 0$ for $0 \leq l \leq 9$

for $j = 1$ to k

$p = integer(max(\mid X_j \mid, \mid Y_j \mid))$

$Q_p = Q_p + 1$

end for;

Output Q_l , for $0 \leq l \leq 9$

2.2 The Multigrid Benchmark (MG)

In this benchmark, a 3-D Poisson PDE $\nabla^2 u = f$ is solved using the multigrid method.

Grid size

$$\Omega_{2^m} = \{x_i = i \cdot h, y_j = j \cdot h, z_k = k \cdot h \mid h = \frac{1}{2^m}, i, j, k = 1, \ldots, 2^m - 1\}$$

Algorithm:

% u,f are 3-D arrays that represent values on the grids.
% The finest grids is Ω_{2^8}
initial f_{2^8}
$u_{2^8} = 0$
repeat 4 times
    $r_{2^8} = f_{2^8} - A(u_{2^8})$ % evalue residual
    $u_{2^8} = u_{2^8} + M(r_{2^8})$ % apply correction
end repeat

% M denotes the V-cycle multigrid operator.
procedure $M(r_{2^k})$
if (k=1)
    $z_{2^1} = S(r_{2^k})$ % apply smoother
    return(z_{2^1})
else
    $r_{2^{k-1}} = P(r_{2^k})$ % project to corser grid
    $z_{2^{k-1}} = M(r_{2^{k-1}})$ % recursive solve
    $z_{2^k} = Q(r_{2^{k-1}})$ % interpolate to finer grid
    $r_{2^k} = r_{2^k} - A(z_{2^k})$ % evalue residual
    $z_{2^k} = z_{2^k} + S(r_{2^k})$ % apply smoother
    return(z_{2^k})
end if

A, S, P, Q are the operators on the grid points.

2.3 The Conjugate Gradient Benchmark (CG)

In this benchmark, the smallest eigenvalue of a matrix is computed using a conjugate gradient method.

Algorithm

Power method
$x = [1, \ldots, 1]$
$\zeta_2 = \zeta_1 = \zeta = it = 0$
Do 15 times
    $it \leftarrow +1$
    Solve the system $Az = x$;
    $\zeta_2 = \zeta_1$
    $\zeta_1 = \zeta$
    $\zeta = max_j |z_j|$
    $x = \zeta^{-1} z$
    Apply Aitken extrapolation $\zeta' = \zeta - \frac{(\zeta - \zeta_1)^2}{\zeta - 2\zeta_1 + zeta_2}$
    Print ζ'
End do

Conjugate gradient method to solve $Az = x$
$z = 0$
$r = x$
$\rho = r^T r$
$p = r$
Do 25 times
    $q = Ap$
    $\alpha = \frac{\rho}{p^T q}$
    $z = z + \alpha p$
    $\rho_0 = \rho$
    $r = r - \alpha q$
    $\rho = r^T r$
    $\beta = \frac{\rho}{\rho_0}$
    $p = r + \beta p$
End do

2.4 The 3-D FFT PDE Benchmark (FT)

In this benchmark, a 3-D PDE is solved using FFTs.

Algorithm

1. Generate $2n_1n_2n_3$ pseudorandom floating point values and fill in the complex array $U_{i,j,k}, 0 \leq i \leq n_1, 0 \leq j \leq n_2, 0 \leq k \leq n_3$

2. % 3-D DFT
   Perform an n_1-point 1D FFT on each of the n_2n_3 complex vectors.
   Then, transpose into $n_2 \times n_3 \times n_1$ complex array.
   Perform an n_2-point 1D FFT on each of the n_3n_1 complex vectors.
   Then, transpose into $n_3 \times n_1 \times n_2$ complex array.
   Perform an n_3-point 1D FFT on each of the n_1n_2 complex vectors.
   Then, transpose into $n_1 \times n_2 \times n_3$ complex array.

3. $\alpha = 10^{-6}, t = 1$
   $W_{i,j,k} = e^{-4\alpha\pi^2(ii^2+jj^2+kk^2)t}V_{i,j,k}$
   where
   $$ii = \begin{cases} i, & \text{for } 0 \leq i \leq n_1/2; \\ i - n_1, & \text{for } n_1/2 \leq i \leq n_1. \end{cases}$$
   jj,kk are similarly defined with n_2 and n_3.

4. Perform inverse 3-D DFT

5. Compute the checksum $\sum_0^{1023} X_{r,s,p}$ where $r = i(mod\, n_1)$, $s = 3i(mod\, n_2)$, $p = 5i$

6. Repeat above for $t = 2, 3, 4, 5, 6$

2.5 The Integer Sort Benchmark (IS)

In this benchmark, a sorting operation is used.

Algorithm

1. Genarate N integer keys $(K_1, K_2, \ldots, K_N)$

2. fot i=1 to 10

   $K_i = i$

   $K_{i+10} = (B_{max} - i)$ % B_{max} is a constant

   end

3. Compute the rank of each key.

4. Verify the result.

2.6 The CFD - lower-upper diagonal Benchmark (LU)

In this benchmark,a CFD computation is represented as a block lower and upper triangular system and is solved using a symmetric successive over-relaxation method.

1. Initialzation
   Set the boundary values of $U_{i,j,k}$ for $(i,j,k) \in \partial D_h$
   $\partial D_h = \{(\xi_i, \eta_j, \zeta_k) : i \in \{1, N_\xi\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : j \in \{1, N_\eta\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : k \in \{1, N_\zeta\}\}$
   Set the initial values of $U^0_{i,j,k}$ for $(i,j,k) \in D_h$
   $D_h = \{(\xi_i, \eta_j, \zeta_k) : 2 \leq i \leq (N_\xi - 1), 2 \leq j \leq N_\eta - 1), 2 \leq k \leq N_\zeta - 1)\}$
   The mesh widths
   $h_\xi = 1/(N_\xi - 1)$; $h_\eta = 1/N_\eta - 1)$; $h_{zeta} = 1/(N_{zeta} - 1)$ with $(N_\xi, N_{eta}, N_{zeta}) \in N$ being the number of mesh points.
   Compute the forcing function vector, $H_{i,j,k}$ for $(i,j,k) \in D_h$

2. Compute the $[RHS]^n_{i,j,k}$ for $(i,j,k) \in D_h$
   RHS $= \{I - \Delta\rho[\frac{\partial(A)^n}{\partial\xi} + \frac{\partial^2(N)^n}{\partial\xi^2}]\} \times \{I - \Delta\rho[\frac{\partial(B)^n}{\partial\eta} + \frac{\partial^2(Q)^n}{\partial\eta^2}]\} \times \{I - \Delta\rho[\frac{\partial(C)^n}{\partial\zeta} + \frac{\partial^2(S)^n}{\partial\zeta^2}]\}\Delta U^n$

3. Form and solve the following systems of linear equations for $[\Delta U_1]_{i,j,k}$ for $(i,j,k) \in D_h$ $\{I - \Delta\tau[D_\xi(A)^n + D^2_\xi(N)^n]\}\Delta U_1 = RHS$

4. Form and solve the following systems of linear equations for $[\Delta U_2]_{i,j,k}$ for $(i,j,k) \in D_h$ $\{I - \Delta\tau[D_\eta(B)^n + D^2_\eta(Q)^n]\}\Delta U_2 = \Delta U_1$

5. Form and solve the following systems of linear equations for $[\Delta U^n]_{i,j,k}$ for $(i,j,k) \in D_h$ $\{I - \Delta\tau[D_\zeta(C)^n + D^2_\zeta(S)^n]\}\Delta U^n = \Delta U_2$

6. Update the solution
   $[U^{n+1}]_{i,j,k} = [U^n]_{i,j,k} + [\Delta U^n]_{i,j,k}$ for $(i,j,k) \in D_h$

7. Repeat steps 2-6 200 times

A, B, C, N, Q, S, H are functions of the synthetic problems.
Steps 2-6 consist of one time-stepping iteration.

2.7 The CFD - scalar penta-diagonal Benchmark (SP)

In this benchmark, a CFD computation is represented as multiple systems of scalar penta-diagonal equations.

1. Initialzation
   Set the boundary values of $U_{i,j,k}$ for $(i,j,k) \in \partial D_h$
   Set the initial values of $U^0_{i,j,k}$ for $(i,j,k) \in D_h$
   Compute the forcing function vector, $H_{i,j,k}$ for $(i,j,k) \in D_h$

2. Compute the $[RHS]^n_{i,j,k}$ for $(i,j,k) \in D_h$

3. Perform the matrix-vector multiplication:
   $[\Delta U_1] = (T_\xi^{-1})^n [RHS]$.

4. Form and solve the following system of linear equations for ΔU_2:
   $\{I - \Delta\tau[D_\xi(\Lambda_\xi)^n] - \Delta\tau[D_\xi^2(\rho(N)^n I)] + \Delta\tau[\epsilon h_\xi^4 D_\xi^4(I)]\}[\Delta U_2] = [\Delta U_1]$
   .

5. Perform the matrix-vector multiplication:
   $[\Delta U_3] = (\overline{N}^{-1})[\Delta U_2]$.

6. Form and solve the following system of linear equations for ΔU_4:
   $\{I - \Delta\tau[D_\eta(\Lambda_\eta)^n] - \Delta\tau[D_\eta^2(\rho(Q)^n I)] + \Delta\tau[\epsilon h_\eta^4 D_\eta^4(I)]\}[\Delta U_4] = [\Delta U_3]$
   .

7. Perform the matrix-vector multiplication:
   $[\Delta U_5] = (\overline{P}^{-1})[\Delta U_4]$.

8. Form and solve the following system of linear equations for ΔU_4:
   $\{I - \Delta\tau[D_\zeta(\Lambda_\zeta)^n] - \Delta\tau[D_\zeta^2(\rho(S)^n I)] + \Delta\tau[\epsilon h_\zeta^4 D_\zeta^4(I)]\}[\Delta U_6] = [\Delta U_5]$.

9. Perform the matrix-vector multiplication:
   $[\Delta U^n] = T_\zeta[\Delta U_6]$.

10. Update the solution
    $[U^{n+1}]_{i,j,k} = [U^n]_{i,j,k} + [\Delta U^n]_{i,j,k}$ for $(i,j,k) \in D_h$

2.8 The CFD - block tridiagonal Banchmark (BT)

In this benchmark,a CFD computation is represented as multiple systems of block tridiagonal equations.

1. Initialzation
   Set the boundary values of $U_{i,j,k}$ for $(i,j,k) \in \partial D_h$
   Set the initial values of $U^0_{i,j,k}$ for $(i,j,k) \in D_h$
   Compute the forcing function vector, $H_{i,j,k}$ for $(i,j,k) \in D_h$

2. Compute the $[RHS]^n_{i,j,k}$ for $(i,j,k) \in D_h$

3. Form and solve the following regular,sparse,block lower triangular system to get $[\Delta U_1]$:
   $(D^n + \omega Y^n)[\Delta U_1] = [RHS]$.

4. Form and solve the following regular, sparse, block upper triangular system to get $[\Delta U^n]$:
   $(I + \omega(D^n)^{-1} Z^n)[\Delta U^n] = [\Delta U_1]$.

5. Update the solution
   $U^{n+1} = U^n + [1/\omega(2-\omega)]\Delta U^n$

3 Preliminary results

Followings are some of the SISAL results.

SUN-4

	size	f77	f77 -O	sisal
Embarresing para.	2^{24}	824.25	805.97	775.13
Multigrid	32^3	45.08	5.47	9.13
Conjugate grdient	1400	75.93	37.04	36.15
3D FFT	64^3	398.26	174.25	209.65
CFD-sp	12^3	202.01	84.40	112.99

f77 -O : Most optimization.

CRAY YMP-C90 (one processor)

	size	f77 -Zv	sisal
Embarresing para.	2^{21}	27.28	20.97
Multigrid	32^3	0.18	1.45

CRAY YMP (one processor)

	size	f77 -Zv	sisal
3D FFT	64^3	11.58	48.77
CFD-sp	12^3	28.24	50.31

CF77 -Zv : Resulting program has maximum vectorization. Specifies use of dependence analyzer fpp before compiling and loading.

Sequent Balance : SISAL on different number of processors

	size	1	2	4	8	16
CFD-sp	12^3	867.86	473.57	289.21	213.71	135.95
FFT	32^3	1184.87	613.43	314.50	167.60	101.64

4 Future research

We are still pursuing experiments with larger datasets. We expect that the experiments with larger datasets will show more speed-up. Previous experiments tend to show that the SISAL language is ideal for the in programming and maintaining of applications for parallel machines because of its implicit parallel semantics. Moreover, our experiments show that, even on a single processor, SISAL codes always perform better than a FORTRAN code that is not fully optimized. With the same effort, applications in SISAL will produce better performance on multiprocessor machines, than those in FORTRAN.

References

[1] David Bailey, John Barton, Thomas Lasinski, and Horst Simon. The NAS Parallel Benchmarks. In *NASA Technical Report RNR-91-002*, January 1991.

5 Appendix

Code examples

Some SISAL codes from the implementation.

- This following code is from the 3-D FFT. It performs the transpose of a vector which is treated as a two dimentional matrix, and executes in parallel.

  If x is a n-vectors by C^n, and $n = rc$, then $x_{r\times c}$ means the matrix $\in C^{r\times c}$ with row = r and column = c , i.e., $[x_{r\times c}]_{kj} = x_{jr+k}$.

  % Perform the transpose $x_{r\times c}$ into $x^T_{r\times c}$

```
function trans(r,c: integer; x: OneD returns OneD)
let
        t2:=
        for j in 0,c-1
            t :=
            for i in 0,r-1
            returns
                    array of x[j+i*c]
            end for;
        returns
                    value of catenate t
        end for;
in
        array_setl(t2,0)
end let
end function
```

- It is difficult to express histogram algorithm in parallel using SISAL 1.x. However, partial histogram only for a range can be expressed easily in parallel. for example, the following sisal codes from the EP calculate the partial histogram.

```
function RamtoL(Ram : DArr returns OneI)
let
        a0,a1,a2,a3,a4,a5,a6,a7:=
        for i in 1, WIDE/2
                ⋮
                elm := (calculation)...
        returns
                value of sum 1 when elm=0
                value of sum 1 when elm=1
                value of sum 1 when elm=2
                value of sum 1 when elm=3
                value of sum 1 when elm=4
                value of sum 1 when elm=5
                value of sum 1 when elm=6
                value of sum 1 when elm>7
        end for;
in
        array OneI[1: a0,a1,a2,a3,a4,a5,a6,a7]
end let
end function
```

- The following code is from the CG. It shows that SISAL codes look like the mathematical equations. Also, to write a program in SISAL, it is best to start from the mathematical point of view.

  Conjugate gradient method

  $z = 0$
  $r = x$
  $\rho = r^T r$
  $p = r$
  Do 25 times
      $q = Ap$
      $\alpha = \frac{\rho}{p^T q}$
      $z = z + \alpha p$
      $\rho_0 = \rho$
      $r = r - \alpha q$
      $\rho = r^T r$
      $\beta = \frac{\rho}{\rho_0}$
      $p = r + \beta p$
  End do

```
% a sparse matrix is represented with Ak and Ck.
% Values stores in Ak and the column locations are in Ck.

function cgsol( nit,n: integer; Ak: TwoD; Ck: TwoI; x: OneD
        returns OneD, double_real )
        for initial
              z := array_fill(1,n,0d0);
              r := x;
              rho := ddot(n,r,r);
              p := x;
              k:= 1;
        while(k¡= nit) repeat
              k:= old k +1;
              q := matvec(n, Ak, Ck, old p);
              alpha := old rho / ddot(n, old p,q);
              z := daxpy( n, alpha, old p, old z);
              rho0 := old rho;
```

```
            n_alpha := -1d0 * alpha;
            r := daxpy( n, n_alpha, q, old r);
            rho := ddot(n, r,r);
            beta := rho / rho0;
            p := daypx( n, beta, r,old p);
        returns
            value of z
        end for
end function
```

- This code block is to multiply a sparse matrix with a vector.

```
% matrix × vector

function matvec( n: integer; Ak :TwoD; Ck: TwoI; Xk: OneD
                 returns OneD)
for i in 1, n
        elm :=
        let
                row := Ak[i];
                rowc := Ck[i];
        in
                for e in row at k
                      s:= e*Xk[rowc[k]];
                returns
                      value of sum s
                end for
        end let;
returns
        array of elm
end for
end function
```