Efficient Parallel Computation of All-Pairs *N*-body Acceleration by Do Loop Folding

FUKUSHIMA, Toshio
(NAOJ)

In the Newtonian celestial mechanics, the all-pairs Nbody acceleration vectors are expressed in the form of double summation as $\mathbf{a}_j = \sum_{k=1,k\neq j}^{N} \mathbf{a}_{jk}$ where the index j runs $1 \le j \le N$. In the ordinary serial computation, the anti-symmetric nature of the mutual accelerations, $\mu_j \mathbf{a}_{jk} + \mu_k \mathbf{a}_{kj} = 0$, is used to halve the total computational amount from $N(N-1)$ to $N(N-1)/2$.

In the parallel computation, however, this property is not always utilized. This means that the total amount of computation is almost doubled. Indeed Figure 1 shows that the speed-up factor of the simple parallelization for various PCs suffers this defects.

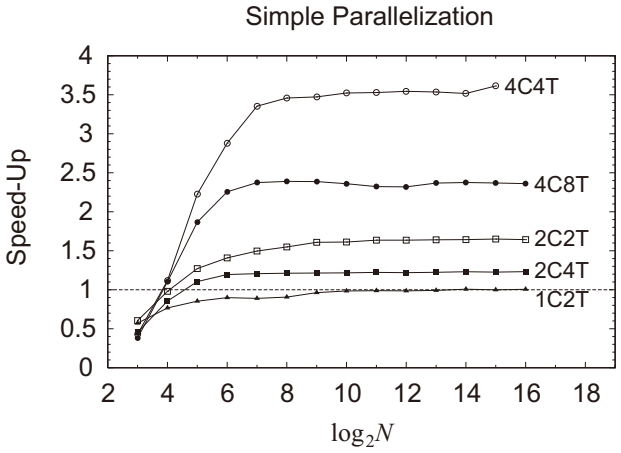

Figure 1: Size Dependence of Speed-Up Factor: Simple Parallelization.

In fact, the observed value is 1.0 for a single-core two-thread processor (1C2T), 1.7 for a dual-core twothread processor (2C2T), 1.2 for a dual-core four-thread processor (2C4T), 3.5 a quad-core four-thread processor (4C4T), and 2.4 for a quad-core 8-thread processor (4C8T), respectively. Meanwhile the ideal value is 1.0 for 1C2T, 2.0 for 2C2T and 2C4T, and 4.0 for 4C4T and 4C8T, respectively. In order to overcome this situation, we apply the idea of folding the outer do loop [1] to the computation of *N*-body acceleration [2].

Consider a do loop of the form

```
do j=1,L {task(j)}
```

where we assume that the task inside the loop is another do loop the length of which is linearly changing with the outer loop index as

```
do k=j+1,L {subtask(j,k)}
```

where the sub task is of an equal computational amount independently on the indices.

Then, we transform this double do loop such that the inner loops contain an equally balanced load as

```
do j=1,(L+1)/2 {task(j);
   if(L+1-j!=j){task(L+1-j)}}
```

As a result, we mostly recover the speed loss in the simple parallel computation of the all-pairs Newtonian *N*-body acceleration vectors as shown in Fig. 2.

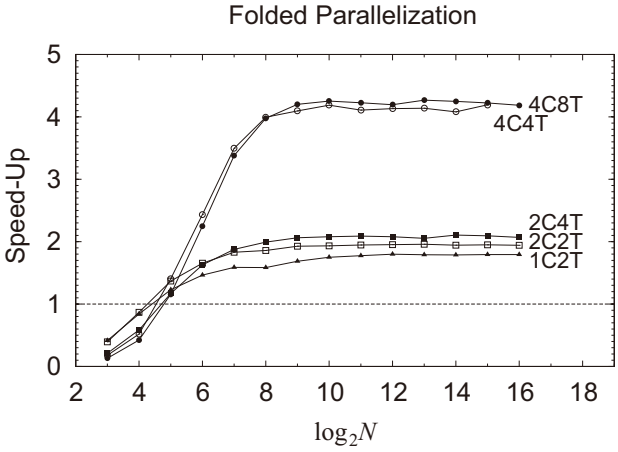

Figure 2: Size Dependence of Speed-Up Factor: Folded Parallelization.

At a consumer PC with a quad-core 8-thread processor for sufficiently large number of *N*, the new program achieves an acceleration factor of 4.5 in the single, 4.2 in the double, and 4.9 in the quadruple precision environments, respectively. This is an interesting result if considering that 4 is the theoretical value for a quad-core processor. We consider that this is due to Intel's Hyper-Threading Technology, which tries to utilize all available computational resource. In general, the present formulation will be applicable to many other computational problems.

References

[1] Ito, T., Fukushima, T.: 1997, *AJ*, **114**, 1260.
[2] Fukushima, T.: 2011, *AJ*, **142**, 18.