

Initially the weighting factor  $W[I]$  is constant for all  $I$ . After each use of the variance function, an Evolve tuple is emitted to cause the object body to execute one loop, in which the contribution of each point to the variance,  $V[I]$ , is used to adjust  $W[I]$ . A point that contributes more would have a smaller  $V_r - V[I]$ . When summed over  $I$ ,  $(V_r - V[I])/Var$  produces  $N - 1$ ; hence,  $(V_r - V[I])/Var(N - 1)$  sums to 1 and satisfies the property of weighting factor. We then combine  $(N - 1)W[I]/N$  with  $(V_r - V[I])/V_r N(N - 1)$  to produce a new weighting factor array.

The idea of evolving function families can also be applied to linear feedback systems and recursive estimation using the Kalman filters, but these are not discussed here.

## 9. Performance Information

Active objects, explained here using BaLinda K, have been given efficient implementations in a BaLinda C++ system operational on a shared memory SUN multi-processor and a distributed memory Fujitsu AP3000 MPP system. The following four tables show the performance data collected for some common object operations and for the Queens program. Noteworthy points are

- (a) While object creation cost is fairly reasonable, in view of the need to attach private tuple spaces and establish object directory entries, but tuple operations on remote objects appear to be rather high; it remains to be seen whether the object search can be shortened and the tuple access speeded up for the most common cases.

Test Case	Performance
Creating a local object	0.54 ms
Creating a remote object	0.77 ms
Object's private tuplespace operation	0.45 ms

- (b) Multi-threaded versions of the program, even when running on a single processor, out performs the sequential version, presumably because of better code and data modularity; in particular, smaller modules of code and data fit into the cache better, thus reducing memory access time.
- (c) The C program, in which all threads share a single tuple space, is considerably worse than the C++ program in which child objects share the

Queens	Solutions	Performance
8	92	0.01
9	352	0.03
10	724	0.14
11	2680	0.69
12	14200	3.83
13	73712	21.90
14	365596	134.82
15	2279184	880.58
16	14772512	6114.38

Queens	1 node	2 node	4 node	8 node
8	0.09	0.10	0.10	0.11
9	0.56	0.33	0.37	0.41
10	1.31	0.71	0.91	0.94
11	6.35	3.16	3.25	3.93
12	36.66	17.75	17.93	20.58
13	194.17	117.73	98.16	104.54
14	968.72	559.94	491.19	516.98
15	6218.84	4089.68	3029.86	3294.04
16	43526.31	26813.03	20526.29	22143.77

tuple space of their parent and tuple operations are distributed; besides reducing critical region contention and tuple search time in smaller hash buckets, a child object residing on the same node as the parent enjoys a shorter object search when accessing the parent's tuple space.

While this is still early days for BaLinda active objects, they seem to be a worthy idea deserving consideration as a software tool for scalable systems.

Queens	1 node	2 node	4 node	8 node
8	0.03	0.02	0.02	0.04
9	0.06	0.04	0.03	0.07
10	0.16	0.08	0.06	0.10
11	0.65	0.32	0.16	0.17
12	3.42	1.67	0.85	0.64
13	19.00	9.30	5.36	4.59
14	105.49	51.20	29.73	17.93
15	676.31	339.68	185.76	111.40
16	4605.01	2252.61	1213.79	838.21

## References

- [1] Z. Xu and K. Hwang, “MPPs and Clusters for Scalable Parallel Computing,” Int. Symp. Parallel Architectures Algorithms Networks, Beijing, June 1996.
- [2] K. Hwang and Z. Xu, “Scalable Parallel Computers: Architecture and Programming,” McGraw-Hill, 1997.
- [3] Special Issue, CACM, Vol. 36, No. 9, 1993.
- [4] J. Chen and G. Zheng, “NDC++: An Approach to Concurrent Extension of C++,” ACM Sigplan Notices, Vol. 32, No. 3, pp. 50–56, 1997.
- [5] D. Kafura, M. Mukherji and G. Lavender, “ACT++: A Class Library for Concurrent Programming in C++ Using Actors,” J. Object-Oriented Programming, Vol. 6, No. 6, pp. 47–55, 1996.
- [6] L. Nigro and F. Tisato, “RTO++: A Framework for Building Hard Real-Time Systems,” J. Object-Oriented Programming, Vol. 6, No. 2, pp. 35–47, 1993.
- [7] D. Boles, “Parallel Object-Oriented Programming with QPC++, Structured Programming,” Vol. 14, No. 4, pp. 158–172, 1993.
- [8] K. Maruyama and N. Raguideau, “Concurrent Object-Oriented Language COOL,” ACM Sigplan Notices, Vol. 29, No. 9, pp. 105–114 (1994).
- [9] S. Matsuoka, K. Taura and A. Yonezawa, “Highly Efficient and Encapsulated Reuse of Synchronization Code in Concurrent Object-Oriented Languages,” ACM Sigplan Notices, OOPSLA, Washington DC, USA, Vol. 28, No. 10, pp. 109–126, September 1993.
- [10] M. D. Feng and C. K. Yuen, “Iterative Computation and Speculative Processing, Software — Concepts and Tools,” Springer-Verlag, Vol. 16, pp. 41–48, 1995.

- [11] C. K. Yuen and M. D. Feng, “*BaLinda Plus, Adding Objects to Parallel Languages, Software — Concepts and Tools,*” Springer-Verlag, Vol. 16, pp. 95–105, 1995.
- [12] C. K. Yuen and M. D. Feng, “*BaLinda — A Simple Parallel Programming Model with Active Objects,*” ISPAN, Taipei, Proceedings published by IEEE Press, pp. 23-29, December 1997.