원소 밀집을 이용한 원소오토마타 모델의 병렬 시뮬레이션

성 영 락†

요 약

본 논문에서는 원소오토마타 모델의 시뮬레이션에서 SIMD형 병렬성을 이용하는 방법을 제안한다. 제안된 방법에서는 SIMD 병렬성을 이용하여 시뮬레이션에 사용되는 컴퓨터 내에 들어 있는 ALU의 이용도를 높이고 시뮬레이션 시간을 줄인다. 그래서 몇 개의 원소들을 결합하여 하나의 표준 크기의 컴퓨터 단어로 만들고 그 원소들의 상태를 동시에 변환시킨다. 제안된 시뮬레이션 방법의 성능을 보이기 위하여, 본 논문에서는 두 가지 원소오토마타 모델을 세 가지 하드웨어 환경에서 시뮬레이션 하였다. 실험결과로부터, 모든 경우에서 시뮬레이션 속도가 매우 크게 향상되었다. 특히 최상의 경우에는 제안된 시뮬레이션 방법에 의한 속도 향상이 20배에 달하는 경우도 있었다.

Parallel Simulation of Cellular Automaton Models using a Cell Packing Scheme

Yeong Rak Seong†

ABSTRACT

This paper proposes a scheme to exploit SIMD parallelism in the simulation of Cellular Automata models. The basic idea is to increase the utilization of an ALU in the underlying computer and to reduce simulation time by exploiting the parallelism. Thus, several cells are packed into a computer word and transit their state together. To show the performance of the proposed simulation scheme, two Cellular Automata models are simulated under three distinct hardware environments. The results show considerably high simulation speed-up for every case. Especially, the simulation speedup with the proposed simulation scheme reaches nearly 20 times in the best case.

1. 서 론

SIMD형 병렬컴퓨터는 하나의 제어 프로세서와 여러 개의 데이터 프로세서로 구성된다. 제어 프로세서는 주기억장치나 하드디스크로부터 제어명령을 읽어 들여 데이터 프로세서들에게 전달하고, 각각의 제어 프로세서는 전달된 제어명령에 따라 자신의 데이터를 처리한다. 일반적으로 이런 유형의 컴퓨터는 매우 크고, 복잡하고, 비싸다는 단점을 가진다.

최근의 몇몇 연구에서 컴퓨터 단어의 길이에 비해 처리되는 데이터의 길이가 짧을 경우 몇 개의 데이터를 묶어서 하나의 컴퓨터 단어로 만들고 이를 병렬로 처리하는 방법을 통하여 SIMD형 병렬성을 하나의 범용 프로세서 내에서 이용하기 위한 방법이 제안되었다. 이 연구들은 크게 하드웨어적인 방법과 소프트웨어적인 방

† 정 회 원 : 국민대학교 전자공학과
논문접수 : 1997년 4월 2일, 심사완료 : 1997년 12월 29일

법으로 나뉜다. 하드웨어적인 방법에서는 분할 가능한 ALU 구조를 개발하고 병렬처리할 데이터들을 각각의 분할된 ALU에 할당하여 처리한다 [1,2]. 최근 들어 인텔사에서는 이를 MMX 기술이라 명명하여 최신의 마이크로프로세서에 이 기능을 탑재하여 출하하고 있다 [3]. 이것을 SIMD형 컴퓨터와 비교해 보면 분할된 ALU는 데이터 프로세서에 대응되고 CPU의 명령 추출부는 제어 프로세서에 대응된다.

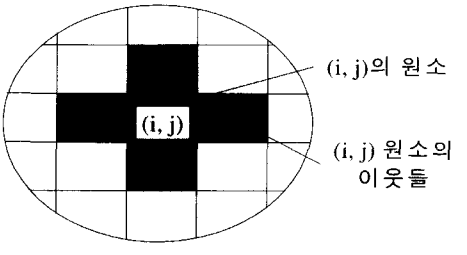
한편, 소프트웨어적인 방법에서는 분할이 불가능한 일반적인 구조의 ALU를 가진 마이크로프로세서내에서 SIMD형 병렬성을 이용하는 것이 연구되었다. 이러한 방법 중에서 가장 널리 알려진 것으로 병렬 디지털 회로 오류 시뮬레이션이 있다 [4]. 병렬 오류 시뮬레이션에서는 하나의 오류를 한 비트로 표현하여 일반적인 순차 컴퓨터 상에서 여러 독립된 오류들을 병렬로 시뮬레이션 한다. 그러므로 동시에 시뮬레이션할 수 있는 오류의 수는 컴퓨터 단어의 길이에 의해 정해진다. 이 방법은 컴퓨터가 부울 논리 연산을 수행할 때, 연산이 각각의 비트 단위가 아니라 단어 단위로 이루어진다는 것을 이용한 것이다. 최근에는 SIMD형 병렬성을 영상처리 분야에 도입하기 위한 연구가 있었다 [5]. 이 연구에서는 하위 수준의 영상처리 알고리즘인 필터링을 SIMD형 병렬성을 이용하여 병렬처리 하였다. 이것은 필터링시에 프로세서내의 ALU의 이용도가 매우 낮음에 착안하였다. 예를 들어 8 비트 영상을 32 혹은 64 비트 ALU를 써서 필터링할 때, 한 화소를 처리하는 데에 필요한 비트의 수는 ALU의 비트 수 보다 훨씬 적다는 것이다. 그래서 이 연구에서는 한 화소를 두 바이트로 표현하고 여러 화소를 한 컴퓨터 단어로 결합시켜 그 화소들을 동시에 처리하였다. 그리고 실험을 통하여 SIMD형 병렬성을 이용하여 하드웨어 비용의 추가 없이 필터링 시간을 크게 줄일 수 있음을 보였다.

본 논문에서는 원소오토마타 모델의 시뮬레이션에 SIMD형 병렬성을 이용하는 방법을 제안한다. 원소오토마타 모델은 동일한 구조를 가진 굉장히 많은 수의 원소로 구성된다. 더욱이 원소오토마타 모델의 시뮬레이션은 근본적으로 많은 반복 계산을 요구하므로 굉장히 많은 시간이 소요된다. 그러므로 많은 연구에서 원소오토마타 모델의 시뮬레이션 시간을 줄이는 방안이 제안되었다 [6-8]. 본 논문에서는 원소오토마타 모델의 시뮬레이션 시간을 줄이기 위하여, 몇 개의 원소들을 묶어서 하나의 컴퓨터 단어로 표현하고 이들을 동시에 시뮬레이션 한다. 그리고 실험을 통하여 제안된 시뮬레이션 방법의 성능을 보인다.

본 논문의 구성은 다음과 같다. 우선 원소오토마타에 대해서 2 절에서 간략하게 소개하고, 3 절에서는 원소오토마타 모델의 시뮬레이션에서 SIMD형 병렬성을 이용하는 방법에 대해 제안한다. 4 절에서는 제안된 시뮬레이션 방법의 성능을 실험적으로 증명한다. 이를 위해서 두 가지 종류의 원소오토마타 모델을 서로 다른 세 워크스테이션 환경에서 실험하고 그 결과를 분석한다. 마지막 5장은 본 논문의 결론이다.

2. 원소오토마타

원소오토마타는 이산 시스템에서 연속(continuous) 시스템의 편미분 방정식에 대응하는 것으로 알려져 있다 [6,9,10]. 원소오토마타는 여러 이산적 성격의 자연계 시스템을 단순하면서도 균일한 방법으로 기술한다. 원소오토마타 모델은 동일한 원소를 기하학적 격자의 격자점에 배치하고 그들을 균일한 형태로 연결하여 구성한다. 각 원소는 동일한 상태변화 규칙을 가지며 동일한 연결형태로 이웃 원소들과 연결되어 있다. 원소오토마타 모델링 방법의 가장 큰 장점은 상태변환규칙과 연결망의 형태가 균일하고 단순하다는 것이다. 이것은 우리가 시스템을 모델링할 때 구성 요소들을 따로 기술하는 것이 아니라 오직 하나의 구성요소에 대한 기술만 하면 된다는 것을 의미한다. 또 원소오토마타는 시스템의 상태를 쉽게 영상으로 보여줄 수 있다는 장점을 가지는데, 이것은 원소오토마타 모델의 공간적인 균일성에 기인한 것이다.



(그림 1) 2 차원 원소오토마타 모델
(Fig. 1) A Two Dimensional Cellular Automata Model

원소오토마타 모델은 임의의 차원으로 무한개의 원

소를 가진다. 그림 1은 2 차원 원소오토마타 모델의 예를 나타낸 것이다. 2 차원 원소오토마타 모델은 다음과 같이 정의된다 [10].

$CA = \langle S, N, T \rangle$: 이산 시간 시스템

여기서

S: 상태의 집합

N: (0,0) 원소의 이웃 원소들과의 연결형태

T: 상태변환함수

로서 다음과 같이 제한된다.

$N \subseteq I^2$

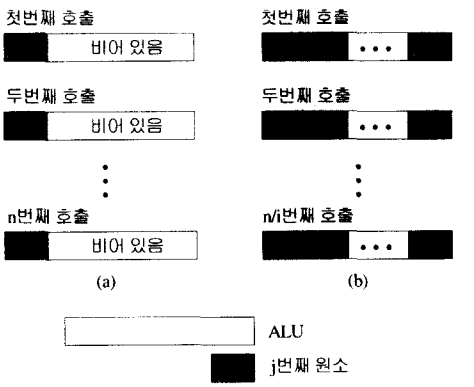
$T: S^{|N|} \rightarrow S$

상기 CA에 의해 정의되는 전체 원소오토마타 시스템을 정의해 보면, Q를 시스템의 전체상태, F를 전체 상태변환함수라고 할 때, $Q = \{q | q: I^2 \rightarrow S\}$로 정의되며 Q의 모든 q에 대해서 $F(q(i,j)) = T(q|N+(i,j))$로 정의된다.

3. 원소 결합과 시뮬레이션

본 절에서는 원소오토마타 모델의 시뮬레이션에서 몇 개의 원소들을 결합하여 한 컴퓨터 단어로 만들어 동시에 시뮬레이션하는 방법을 제안한다. $x \times y$개의 원소로 이루어진 원소오토마타 모델을 8 비트, 16 비트, 32 비트 컴퓨터를 이용하여 시뮬레이션하는 경우를 생각해보자. 또한 한 원소의 상태는 ON/OFF 두 가지의 상태를 가진다고 가정하자. 전통적인 원소오토마타 모델의 시뮬레이션 방법에서는 한번에 한 원소씩 상태 변환을 일으키므로, 전체 시스템이 한번의 상태변환을 하기 위해선 상태변환함수가 시뮬레이션하는 컴퓨터의 ALU의 길이와는 상관없이 $x \times y$번 만큼 호출되어야 한다. 그러나 한 원소는 한 비트만으로 표현되므로 ALU의 길이에 비해 시뮬레이션에 실제로 이용되는 비트의 수는 매우 작다. 다시 말해서 ALU의 길이가 길수록 ALU의 이용도는 떨어지게 된다.

본 논문에서는 원소오토마타 모델의 시뮬레이션에서 여러 개의 원소들을 하나로 묶어서 한꺼번에 상태변환



(그림 2) 상태변환함수가 호출될 때 ALU에서 처리되는 원소 (a) 전통적인 방법 (b) 원소 결합 방법
(Fig. 2) Cells in ALU While Executing State Transition Function (a) Traditional Scheme (b) Cell Packing Scheme

을 시킴으로써, 상태변환함수의 호출 횟수를 줄이고 ALU의 이용도를 늘이는 방법을 제안한다. 그림 2는 이것을 그림으로 나타낸 것이다. 긴 직사각형내에 채색된 짧은 직사각형들이 들어 있는 것은 상태변환함수가 호출되었을 때 ALU 내에서 처리되는 원소들을 나타낸 것이다. 그림 2(a)는 전통적인 시뮬레이션 방법을 이용하는 경우로서 한번에 한 원소씩 다음 상태를 계산한다. 그러므로 n 개의 원소가 상태변환을 하기 위해서는 상태변환함수가 n 번 호출되어야 한다. 반면 그림 2(b)는 본 논문에서 제안하는 방법을 이용하는 경우로서 한번에 i 개의 원소들이 상태변환을 일으키므로 n/i 번의 상태변환함수의 호출로 n 개의 원소들의 다음 상태가 계산된다.

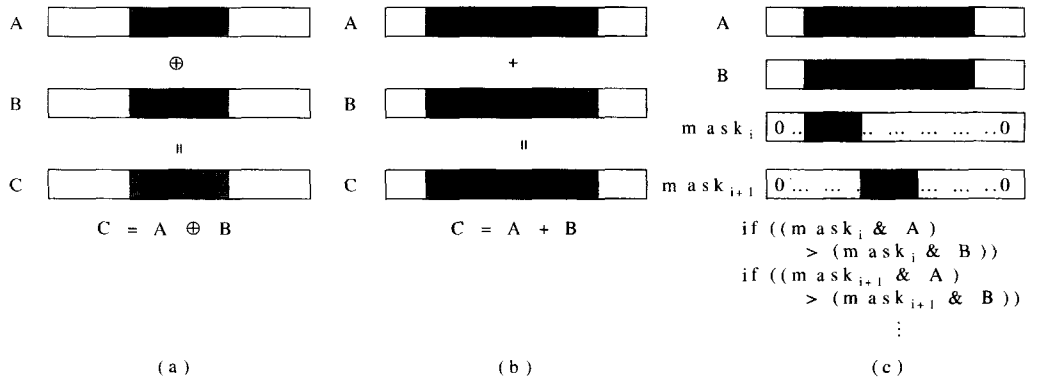
이와 유사한 문제들이 영상처리나 최근 각광받고 있는 멀티미디어 응용 프로그램 개발 과정에서 발생한다. 현재 시장에 소위 MMX 기술이라는 이름으로 출시되고 있는 인텔사의 마이크로프로세서들은 이러한 문제들을 해결하기 위하여 하드웨어적인 방법을 이용한다 [3]. 즉 32 비트 ALU를 한 바이트 또는 두 바이트 단위로 분할하여 각각의 분할된 ALU에 병렬로 데이터를 가져와서 같은 연산을 분할시켜 수행시키고 있다. 본 논문에서는 이와는 달리 소프트웨어적인 방법을 제안한다. 즉 MMX 기술을 채용하지 않은 일반적인 프로세서를 이용하여 MMX 프로세서와 유사한 방법으로 병렬로 데이터를 처리하는 것이다. 최근 이러한 방법을 병

럴 영상처리 문제에 응용한 예가 있다 [5]. 이 연구에서는 몇 개의 화소를 결합하여 하나의 컴퓨터 단어로 만들고 동시에 처리하는 방법을 통하여 ALU의 이용도를 증가 시키고 전체 처리시간을 줄였다. 이 연구에서는 영상처리에서 기본적인 데이터의 단위가 화소이므로 한 컴퓨터 단어로 결합되는 단위 데이터는 최소한 한 바이트 이상 차지한다. 그래서 이 연구에서 제안된 방법은 MMX 컴퓨터에서와 마찬가지로 처리되는 단위 데이터가 최소한 한 바이트 이상의 길이를 가질 경우에 적용될 수 있다.

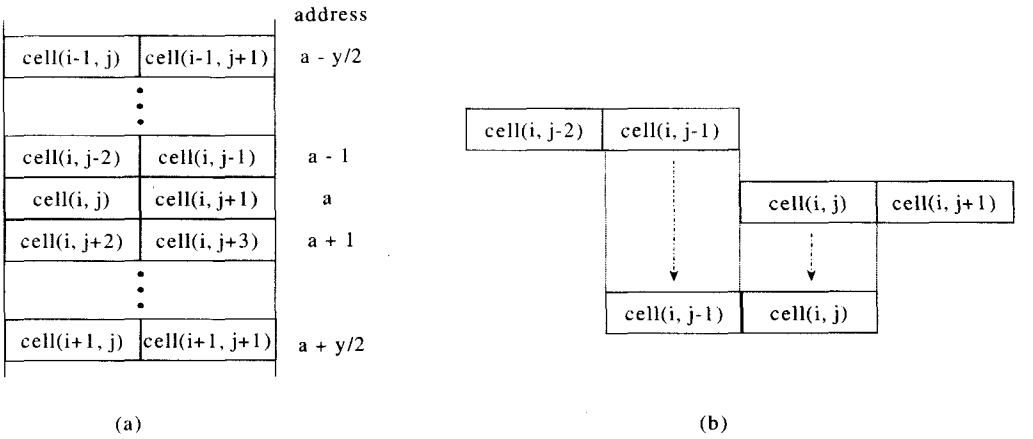
본 논문에서는 이 방법을 발전시켜 원소오토마타 모델의 시뮬레이션에 도입한다. 그래서 몇 개의 원소들을 결합하여 한 단어로 표현하고 그들의 상태를 동시에 변환시킨다. 이런 시뮬레이션 방법은 상태변환함수의 호출 횟수를 줄이고 ALU의 이용도를 높여서 시뮬레이션 시간을 감소시킬 것이다. 이것을 일반적인 MMX 컴퓨터나 [5]의 연구와 비교해 보면 본 논문에서 다루는 원소오토마타는 각 원소의 상태의 경우의 수가 매우 적으므로 한 원소의 상태를 표시하기 위해 소요되는 기억장치의 길이는 한 바이트보다 짧다. 그러므로 각 바이트별로 병렬 처리가 이루어 지는 것이 아니라 수 비트별로 혹은 극단적인 경우에는 한 비트별로 병렬처리가 이루어져야 한다.

일반적인 ALU 구조에서 이 방법을 이용하기 위해선 두 가지의 문제가 생긴다. 첫째는 상태변환함수를 어떻게 한 단어로 결합된 각각의 원소에게 적용하는가 하는 것이다. 상태변환함수는 일련의 ALU 연산에 의해 정의된다. 본 논문에서 제안하는 시뮬레이션 방법의 관점에서, ALU 연산은 크게 두 개의 그룹으로 구분할 수 있다. 첫 번째 그룹은 결합된 각각의 원소에게 연산이 적용될 수 있는 것들이고 두 번째 그룹은 그렇지 않은 연산들이다. 첫 번째 그룹에는 모든 비트별(bitwise) 연산, 자리 이동 연산, 정수 더하기, 정수 빼기, 정수 곱하기 등의 연산이 포함된다. 비트별 연산은 언제나 SIMD형 병렬성을 이용할 수 있으며 정수 더하기, 정수 빼기 등의 연산은 오버플로우(overflow)나 언더플로우(underflow)가 발생하지 않을 경우에 SIMD형 병렬성을 이용할 수 있다. 두 번째 그룹은 조건 연산, 비교 연산, 부동 소수점 연산 등 다른 모든 연산을 포함한다. 이들 연산이 상태변환함수에 포함될 경우 이들 연산에 대해서는 병렬처리가 되지 않고 한 단어내에 결합된 원소들에 대해 각각 순차적으로 연산을 수행해야 한다. 그러므로 두번째 그룹의 연산이 포함된 원소오토마타 모델을 본 논문에서 제안하는 방법을 이용하여 시뮬레이션할 경우 그 연산을 수행하기 위해서는 병렬성을 제안할 것이다

그림 3은 첫번째 그룹 및 두번째 그룹 연산들이 이루어 지는 예이다. 그림 2와 마찬가지로 긴 직사각형은 ALU를 나타내는 것이고 각각의 작은 채색된 직사각형은 하나의 원소를 나타낸 것이다. 그림 3(a)와 그림 3(b)는 첫번째 그룹의 연산들이다. 앞서 언급한 바와 같이 그림 3(a)의 XOR 연산은 비트별 연산이므로 ALU에 할당된 여러 원소들이 동시에 연산을 수행하여도 이웃의 다른 원소들에게 영향을 주지 않는다. 그리고 그림 3(b)의 더하기 연산은 오버플로우가 발생하지 않는 경우 역시 동시에 계산될 수 있다. 그러나 만약



(그림 3) SIMD 병렬성과 여러 연산 (a) XOR 연산 (b) 더하기 연산 (c) 조건 연산
(Fig. 3) SIMD Parallelism and Operations (a) XOR Operation (b) Add Operation (c) Conditional Operation



(그림 4) 이웃 원소에 대한 동시 접근 (a) 주 기억장치상의 원소의 배열 (b) 연속된 원소들의 결합
(Fig. 4) Simultaneous Access to Neighbor Cells (a) Cells in Main Memory (b) Packing of Adjacent Cells

오버플로우가 발생하면 한 원소의 계산에서 발생된 자리 올림이 다른 원소에게 전달되어 영향을 주므로 동시에 계산될 수 없다. 그림 3(c)는 각각의 비트별로 혹은 여러 비트의 그룹별로 크기를 비교하는 명령어가 없으므로 각 원소를 따로 마스트하여 크기를 비교하여야 한다. 즉, 계산이 동시에 처리되지 못하고 그림처럼 순차적으로 이루어져야 한다.

두 번째는 다른 주소에 존재하는 원소들을 동시에 접근하기 위한 방법이다. 같은 단어로 결합된 원소들의 상태를 동시에 변환시키기 위해선 그들의 이웃 원소들이 또한 적당하게 결합되어 있어서 동시에 접근될 수 있어야 한다. 그러나 많은 경우에 동시에 접근되어야 할 이웃 원소들이 같은 단어에 결합되어 있지 않다. 예를 들어, 시뮬레이션 환경이 8 비트 컴퓨터라고 가정하자. 그리고 시뮬레이션될 원소오토마타 모델은 $x \times y$ 개의 원소들로 구성되며, 각 원소는 4 비트로 표현되며 동서남북 방향의 원소들 및 자기 자신을 포함하여 5개의 이웃 원소모델들을 가진다고 하자. 그러므로 한 컴퓨터 단어는 바이트이며 제안된 방법에 의하면 두 개의 원소들이 결합되어 동시에 처리될 수 있다. 그림 4(a)는 주 기억장치 상에 원소들이 분포하는 상태를 나타낸 것이다. 여기서 a번지에 저장된 원소 (i,j)와 $(i,j+1)$의 상태변환을 생각해보자. 우선 전통적인 방법으로 시뮬레이션할 경우 이 원소오토마타 모델의 상태변환함수를 T()라고 할 때 이들 두 원소의 다음 상태는 다음과 같이 주어진다.

$$S_{i,j}' = T(S_{i-1,j}, S_{i,j-1}, S_{i,j}, S_{i,j+1}, S_{i+1,j})$$

$$S_{i,j+1}' = T(S_{i-1,j+1}, S_{i,j}, S_{i,j+1}, S_{i,j+2}, S_{i+1,j+1})$$

여기서 $S_{i,j}$와 $S_{i,j}'$는 원소 (i,j)의 현재 상태와 다음 상태를 의미한다. 그리고 원소를 결합한 모델에 대한 상태변환함수를 G()이라고 하면 원소 (i,j)와 $(i,j+1)$의 다음 상태는 다음과 같이 주어진다.

$$\langle S_{i,j}', S_{i,j+1}' \rangle = G(\langle S_{i-1,j}, S_{i-1,j+1} \rangle, \langle S_{i,j-1}, S_{i,j} \rangle \langle S_{i,j}, S_{i,j+1} \rangle, \langle S_{i,j+1}, S_{i,j+2} \rangle, \langle S_{i+1,j}, S_{i+1,j+1} \rangle)$$

여기서 중요한 것은 〈〉로 묶어진 원소들은 동시에 접근될 수 있어야 한다는 것이다. 그러나 그림에서 보면 이들 중에는 다른 주소에 할당된 원소들도 있다. 예를 들어, 원소 (i, j-1)과 (i, j)를 동시에 접근 할 수 있어야 하지만 그 두 원소들은 같은 주소상에 존재하지 않으므로 추가적인 연산이 없이는 동시에 접근할 수 없다. 이런 경우 본 논문에서는 그림 4(b)에 나타난 것처럼, 자리 이동 연산과 비트별 OR 연산을 이용하여 그러한 원소들을 하나로 결합한다. 이 방법을 이용하여 동

시에 읽혀져야 할 모든 셀들을 동시에 접근할 수 있다.

본 논문에서 제안한 방법에 의해 한 단어 내에 결합되는 원소의 수 ρ는 다음과 같이 정의된다.

$$\rho = \frac{\text{시뮬레이션하는 ALU의 길이}}{\text{한 세포를 시뮬레이션하는 데에 필요한 비트 수}}$$

ρ는 본 논문에서 제안하는 시뮬레이션 방법을 이용할 때의 시뮬레이션 속도 향상의 이론적인 상한선이다. SIMD형 병렬컴퓨터에서 각 데이터 프로세서는 상태변환 중간과정에서의 계산 값을 저장하고 있어야 한다. 그러므로 본 논문에서 제안한 방법으로 원소오토마타 모델을 시뮬레이션하기 위해서는 한 원소를 시뮬레이션하기 위하여 할당된 ALU의 각 부분들도 중간 시뮬레이션 결과를 저장하고 있어야 한다. 많은 원소오토마타 모델들에서 원소의 상태를 표현하는 데에 필요한 비트의 수는 중간 시뮬레이션 결과를 저장할 때 필요한 비트의 수와는 다르다. 결국 시뮬레이션하는 데에 필요한 비트의 수는 시뮬레이션되는 원소의 상태를 표현하는 데에 필요한 비트 수 외에도 원소오토마타 시뮬레이션의 중간 결과를 저장하는 데에 필요한 비트 수에 의해 결정된다.

4. 실 험

본 절에서는 본 논문에서 제안하는 시뮬레이션 방법의 성능을 평가하기 위해 두 가지의 원소오토마타 모델들을 기존의 방법과 제안된 방법으로 시뮬레이션하였다. 첫 번째는 생명게임(game of "life") 모델이다. 이 모델은 1970년 John Conway에 의해 제안된 이래 전 세계의 많은 과학자들로부터 주목 받았던 모델이다[11]. 원래 이 모델은 2 차원 원소오토마타 모델로서 다음과 같이 정의된다.

$$CA_{life} = \langle S, N, T \rangle$$

여기서

$$S = \{ALIVE, DEAD\}$$

$$N = \{(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)\}$$

$$s'_{i,j} = s_{i,j}(t+1)$$

$$= T(s_{i-1,j-1}, s_{i-1,j}, s_{i-1,j+1}, s_{i,j-1}, s_{i,j}, s_{i,j+1}, s_{i+1,j-1}, s_{i+1,j}, s_{i+1,j+1})$$

$$= LIFE(s_{i-1,j-1}, s_{i-1,j}, s_{i-1,j+1}, s_{i,j-1}, s_{i,j}, s_{i,j+1}, s_{i+1,j-1}, s_{i+1,j}, s_{i+1,j+1})$$

$LIFE\ (s_{i-1,j-1}, s_{i-1,j}, s_{i-1,j+1}, s_{i,j-1}, s_{i,j}, s_{i,j+1}, s_{i+1,j-1}, s_{i+1,j}, s_{i+1,j+1})$

※ 시간 t에서, 만약 원소 (i,j)가 생존상태이면, $s_{i,j}(t) = 1$;

그렇지 않으면, $s_{i,j}(t) = 0$.

$sum \leftarrow s_{i-1,j-1}(t) + s_{i-1,j}(t) + s_{i-1,j+1}(t) + s_{i,j-1}(t) + s_{i,j+1}(t) + s_{i+1,j-1}(t) + s_{i+1,j}(t) + s_{i+1,j+1}(t)$

```
If s_{i,j}(t) Then
    If sum = 2 or sum = 3 Then
        s_{i,j}(t+1) ← 1
    Else
        s_{i,j}(t+1) ← 0
    End If
Else
    If sum = 3 Then
        s_{i,j}(t+1) ← 1
    Else
        s_{i,j}(t+1) ← 0
    End If
End If
```

(그림 5) 생명게임
(Fig. 5) Game of "Life"

여기서 LIFE는 그림 5과 같이 정의된다. LIFE 함수는 [11]에서 정의된 것으로 1) 주변에 2, 3 곳에 생명체들이 살아 있으면 그곳에는 생명체들이 계속해서 생존하고, 2) 주변에 4 곳 이상에 생명체들이 살면 그곳은 너무 복잡해서 그곳의 생명체는 소멸하고, 3) 주변에 정확하게 3 곳에 생명체들이 살면 새로운 생명체

가 발생한다는 사실을 식으로 나타낸 것이다. LIFE 함수는 조건 연산을 포함하고 있으므로, 제안된 시뮬레이션 방법을 이용하기 위해선 조건 연산을 순차적으로 처리하기 위한 오버헤드가 추가되어야 한다. 2 차원 생명게임 모델은 이웃 원소의 수가 9 개이므로 추가된 오버헤드에 비하여 병렬성이 적다. 그래서 본 논문에서는 생명게임 모델을 3 차원 원소오토마타 모델로 확장하여 SIMD형 병렬성을 늘이고 조건 연산에 의해 야기된 시뮬레이션 오버헤드의 영향을 줄였다. 결국 각 원소는 27 개의 이웃을 가진다. 각 원소의 상태를 표시하기 위해서는 오직 한 비트만 필요하지만, 상태변환을 위한 중간 결과 값인 "sum"이 0에서 26까지의 범위를 가지므로 제안된 시뮬레이션 방법으로 각 원소들을 시뮬레이션하기 위해서는 5 비트가 필요하다. 그러나 본 논문에서는 문제를 단순화시키기 위하여 각 원소를 한 바이트로 표현하였다. 시뮬레이션 모델은 $1024\times1024\times3$ 개의 원소들로 구성하였다.

두 번째 모델로는 패리티 모델을 도입하였다. 이 모델은 원소오토마타의 도입기에 Edward Fredkin에 의해 소개되었다 [6]. 이 모델에서 각 원소는 5 개의 이웃을 가지며 어떤 원소의 다음 상태는 이웃 원소의 패리티에 의해 결정된다. 즉, 각 원소는 그 이웃 원소 중에 생존 원소들의 수가 홀수인지 짝수인지에 의해 생존 혹은 사망 원소로 된다. 패리티 모델은 다음과 같이 정의된다.

$$CA_{parity} = \langle S, N, T \rangle$$

여기서

$$S = \{ALIVE, DEAD\}$$

$$N = \{(-1,0), (0,-1), (0,0), (0,1), (1,0)\}$$

$$s'_{i,j} = s_{i,j(t+1)}$$

$$= T(s_{i-1,j},\ s_{i,j-1},\ s_{i,j},\ s_{i,j+1},\ s_{i+1,j})$$

$$= s_{i-1,j} \oplus s_{i,j-1} \oplus s_{i,j} \oplus s_{i,j+1} \oplus s_{i+1,j}$$

이 상태변환 규칙의 경우 오직 비트별 연산만이 필요하므로 제안된 시뮬레이션 방법에서는 한 단어로 결합된 모든 원소들이 동시에 상태변환을 수행할 수 있다. 그리고 상태변화의 중간값을 표현하기 위해 필요한 비트수도 원소의 상태를 표시하기 위해 필요한 비트수와 같이 한 비트이다. 본 논문에서는 1024×1024 개의 원소들에 대해서 시뮬레이션하였다.

〈표 1〉 시뮬레이션 환경
〈Table 1〉 Simulation Environments

	워크스테이션명	프로세서	운영체제
환경 I	DEC-3000 M600 [12]	64 비트 Alpha	OSF1
환경 II	Sun Sparc II [13]	32 비트 Sparc	Solaris
환경 III	IBMPC Pentium [14]	32 비트 Pentium	Linux

이 두 원소오토마타 모델을 표 1의 세 상용 워크스테이션 환경에서 전통적인 원소오토마타 시뮬레이션 방법과 제안된 원소오토마타 시뮬레이션 방법으로 시뮬레이션하였다. 본 실험에서는 모든 경우에 대해서 전체 상태를 한번씩 변환시켰다. 즉 주어진 원소오토마타 시스템 내의 모든 원소들이 한번씩 상태변환을 일으켰다. 그러나 실제적인 시뮬레이션에서는 이런 상태변환이 수백 번씩 반복될 것이고 원소의 수 또한 훨씬 더 많을 것이므로 시뮬레이션에 굉장히 긴 시간이 소요될 것이다. 본 실험에서 한번씩만 상태 변환을 한 것은 원소오토마타 시뮬레이션은 매우 정형화되어 있어서, 시뮬레이션을 반복하는 횟수가 증가하면 전체 시뮬레이션 시간이 선형적으로 증가하기 때문에, 여러번 반복하여 평균 수행시간을 구할 필요없이 한번 수행할 때의 시뮬레이션 시간만으로 성능의 비교가 가능하기 때문이다.

표 2는 시뮬레이션 결과를 나타낸 것이다. 모든 경우에서 제안된 시뮬레이션 방법을 사용한 경우에 전체 수행시간이 감소되었다. 특히 환경 I에서의 시뮬레이션 속도향상이 크게 나타나는데 이것은 환경 I이 환경 II나 III과는 달리 64 비트 환경이기 때문이다. 그러나 환경 II와 환경 III을 비교해 보면 어느 한쪽이 다른 쪽보다 우월하게 나타나지는 않았다. 3 차원 생명게임 모델의 경우에는 환경 II가 환경 III에 비해 우수했으나, 패리티 모델의 경우에는 반대의 양상이 나타났다. 이것은 Sparc 프로세서와 Pentium 프로세서의 내부 구조의 차이에서 기인한 것으로 판단된다.

앞서 언급했듯이 제안된 시뮬레이션 방법에서는 두 가지의 SIMD 병렬성을 제안하는 요소가 있다. 첫째는

〈표 2〉 실험 결과
〈Table 2〉 Experimental Results

(단위: 초)

모델 / 환경	생명게임				패리티			
	T_c	T_t	ρ	T_c/T_t	T_c	T_t	ρ	T_c/T_t
환경 I	17.804	2.967	8	6.034	8.972	0.460	64	19.504
환경 II	45.155	12.233	4	3.718	15.239	2.156	32	7.068
환경 III	22.360	6.954	4	3.215	12.870	1.271	32	10.126

T_c: 전통적인 시뮬레이션 방법에 의한 시뮬레이션 시간
T_t: 제안된 시뮬레이션 방법에 의한 시뮬레이션 시간

결합된 각 원소의 상태변환에 골고루 적용될 수 없는 연산에 의한 오버헤드이고, 둘째는 이웃한 원소들을 동시에 접근하기 위해 부가되는 자리 이동 및 비트별 OR 연산에 의한 오버헤드이다.

3 차원 생명게임 원소오토마타 모델의 시뮬레이션의 경우, 이웃 원소의 수가 27개나 되므로 내재한 SIMD 병렬성이 매우 크다. 그러므로 상태변환함수 내에 병렬화가 불가능한 연산을 포함하고 있지만 속도향상은 이론적인 상한선인 ρ와 비교해 볼 때 매우 근접하게 나타났다. 한편 패리티 모델의 경우 상태변환에 필요한 모든 연산이 비트별 연산이므로 생명게임 모델에 비하여 시뮬레이션 오버헤드가 적다. 그러나 이웃 원소들의 수가 적어서 시뮬레이션에 내재된 병렬성은 적은 편이다. 시뮬레이션 결과에서도 패리티 모델의 시뮬레이션 속도 향상은 ρ에 비해서 작게 나타났다. 이것은 제안된 시뮬레이션 방법에 의해 추가되는 오버헤드는 내재한 병렬성에 비해 시뮬레이션 속도향상에 적은 영향을 미친다는 것을 의미한다. 그럼에도 불구하고 패리티 모델 시뮬레이션은 수행 환경에 상관없이, 모든 하드웨어 환경에서 매우 높은 속도향상을 보였다. 특히, 환경 I의 경우 속도향상이 거의 20 배에 달했다. 이 정도의 속도향상은 일반적인 병렬처리 기술을 이용할 경우 수십 개의 프로세서를 이용해야 가능한 것이다. 그러나 본 논문에서 제안된 시뮬레이션 방법을 이용할 경우, 하나의 프로세서만을 이용하여 동등한 속도 향상을 얻을 수 있었다. 이 결과로 볼 때, 원소오토마타 모델의 시뮬레이션에 내재한 SIMD형 병렬성이 매우 크며, 본 논문에서 제안된 시뮬레이션 방법은 그것을 효과적으로 잘 이용함을 알 수 있다.

5. 결 론

본 논문에서는 분할 불가능한 구조의 일반적인 ALU를 가진 마이크로프로세서 상에서 원소오토마타 모델의 시뮬레이션에 SIMD형 병렬성을 도입하였다. 이를 위하여 몇 개의 원소들을 결합하여 컴퓨터 단어로 만들고 그 원소들의 상태를 동시에 변환시켰다. 이것은 하드웨어를 이용한 MMX 방식의 마이크로프로세서에서 제시된 명령어들을 이용하여 데이터를 처리하는 것에 비하여 ALU를 더 작은 단위로 분할하여 시뮬레이션하는 것을 가능하게 해 준다. 제안된 시뮬레이션 방법의 성능을 보이기 위하여 두 개의 널리 알려진 원소오토마타 모델을 세 가지의 상업용 워크스테이션 환경에서 시뮬레이션하였다. 실험 결과 우리는 원소오토마타 모델의 시뮬레이션에서 SIMD형 병렬성이 매우 크다는 것과, 본 논문에서 제안된 시뮬레이션 방법이 SIMD형 병렬성을 잘 이용하고 있다는 것을 알 수 있었다. 특히, DEC의 Alpha 워크스테이션 환경에서 패리티 원소오토마타 모델을 시뮬레이션한 결과, 제안된 시뮬레이션 방법을 이용하면 일반적인 시뮬레이션 방법에 비해 시뮬레이션 속도가 거의 20배나 향상되는 것을 보였다. 이러한 정도의 속도 향상은 거의 20~40개의 프로세서로 구성된 병렬처리 컴퓨터를 이용하여야만 얻을 수 있는 높은 수치이다. 그러나 본 논문에서 제안된 알고리즘을 이용할 경우 일반적인 순차적 프로세서 하나만을 이용하여 동등한 시뮬레이션 속도 향상을 얻을 수 있었다.

참 고 문 헌

[1] K. Diefendorff, M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE Micro*, vol. 12, no. 2, pp. 40-63, Apr. 1992.

[2] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, vol. 15, no. 2, pp. 22-32, Apr. 1995.

[3] Intel Corporation, *The complete guide to MMX™ Technology*, McGraw-Hill, 1997.

[4] G. Russell, D. J. Kinniment, M. R. McLauchlan, E. G. Chester, *CAD for VLSI*, Van Nostrand Reinhold (UK), 1985.

[5] Y. R. Seong, T. G. Kim, K. H. Park, "Packing Scheme for Mean- Filtering of an 8-bit Image," *Electronics Letters*, vol. 32, no. 1, pp. 29-30, Jan. 1996.

[6] T. Toffoli, N. Margolus, *Cellular Automata Machines*, MIT Press, 1987.

[7] M. Cannataro, S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, D. Talia, "A Parallel Cellular Automata Environment on Multicomputers for Computational Science," *Parallel Computing*, vol. 21, np. 5, pp. 803-824.

[8] C. Di Napoli, M. Giordano, M. Mango Furnari, R. Napolitano, "A Portable Parallel Environment for Complex Systems Simulation Through Cellular Automata Networks," *J. Systems Architecture*, vol. 42, no: 5, pp.341-348.

[9] B. P. Zeigler, *Theory of Modelling and Simulation*, John Wiley, 1976.

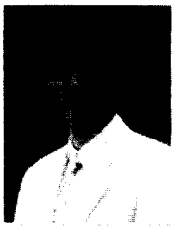
[10] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984.

[11] M. Gardner, "The Fantastic Combinations of John Conway's New Solitaire Game `Life'," *Scientific American*, vol. 23, no. 4, pp. 120-123, Apr. 1970.

[12] DEC, *Alpha Architecture Handbook*, 1992.

[13] SPARC International, *The SPARC Architecture Manual*, 1992.

[14] D. Anderson, T. Shanley, *Pentium Processor™ System Architecture*, MindShare, Inc., 2nd ed., 1995.



성 영 락

1989년 한양대학교 전자공학과, 공학사

1991년 한국과학기술원 전기 및 전자공학과, 공학석사

1995년 한국과학기술원 전기 및 전자공학과, 공학박사

1995년~1996년 한국과학기술원 위촉연구원,

1996년~1998년 국민대학교 전임강사

1998년~현재 국민대학교 조교수

관심분야 : 멀티미디어 시스템, 시스템 모델링 및 시뮬레이션, 병렬처리