데이터 중첩을 통한 페트리네트의 병렬 시뮬레이션

김영찬*, 김탁곤**

Parallel Simulation of Bounded Petri Nets using Data Packing Scheme

Young Chan KIM, Tag Gon KIM

Abstract

This paper proposes a parallel simulation algorithm for bounded Petri nets in a single processor, which exploits the SIMD(Single Instruction Multiple Data)-type parallelism. The proposed algorithm is based on a data packing scheme which packs multiple bytes data in a single register, thereby being manipulated simultaneously. The parallelism can reduce simulation time of bounded Petri nets in a single processor environment. The effectiveness of the algorithm is demonstrated by presenting speed-up of simulation time for two bounded Petri nets.

Key Words: Petri Nets, Parallel Simulation, SIMD, Data Packing

* 한밭대학교 정보통신-컴퓨터공학부

** 한국과학기술원 전자전산학과

1. 서론

반도체 기술이 빠르게 발전함으로서 알파 프로세서, 펜티엄 프로세서 [1] 혹은 스팍 프로세서 [2]와 같은 32비트나 64비트 마이크로프로세서들을 우리의 주변 환경에서 흔히 접하게 되었다. 하지만, 많은 응용프로그램에서 이러한 프로세서들의 증가된 하드웨어 능력을 충분히 사용하지 못하고 있다. 한 가지 예는, 한정 페트리네트(Bounded Petri Nets)의 시뮬레이션에 32나 64비트 프로세서를 사용하는 것이다.

이 경우 한정 페트리네트의 마킹 원소를 나타내기 위해 필요로 하는 비트 수는 프로세서 레지스트의 비트 수보다 매우 적을 것이다. 프로세서의 활용도를 높이기 위해서 여러 개의 마킹(marking) 원소들을 하나의 레지스트에 중첩시키고 동시에 처리하는 것이 바람직하다. 이와 같이 할 수 있기 위해서는 페트리네트의 시뮬레이션 알고리즘이 병렬화 되어야 한다. 다행히 페트리네트의 시뮬레이션 알고리즘은 음수가 아닌 자연수의 더하기, 빼기와 비교연산만이 필요하므로 병렬화가 가능하다. 이러한 아이디어를 기반으로 해서 저자들은 하나의 프로세서 상에 페트리네트를 병렬 처리할 수 있는 SIMD(Single Instruction Multiple Data)-타입의 알고리즘을 제안한다.

기존 병렬 시뮬레이션 연구들은 시간페트리네트에 대해 병렬/분산시뮬레이션에 집중적으로 이루어왔다 [5,6,7]. 이들은 보통의 페트리네트를 다루는 본 논문과는 다르게 시간페트리네트를 대상으로 한다. 또한 프로세서가 하나인 보통의 컴퓨터상에서 시뮬레이션을 병렬화를 시도하는 본 논문과는 다르게, 병렬 아키텍처를 가지는 컴퓨터를 대상으로 하는 병렬/분산 알고리즘을 다루고 있다.

이 논문의 구조는 다음과 같다. 2절에서는 이 논문을 이해하기 위해 필요한 기본적인 정의와 결과들을 설명한다. 3절에서는 데이터 중첩 방법을 제시한다. 4절에서는 데이터 중첩 방법에 기반한 병렬 시뮬레이션 알고리즘을 제시한다. 5절에서는 제시된 알고리즘의 효율성을 보이기 위해 실험을 실시하고 그 결과를 보여준다. 6절에서 이 논문의 결론을 제시한다.

$$TPN_1 = (P, T, I, O, \mu)$$
$$P = \{p_1, p_2, p_3, p_4\}$$
$$T = \{t_1, t_2, t_3, t_4\}$$
$$\mu = (1,2,0,0)$$
$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} \qquad D^+ = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 0 & 0 \end{pmatrix}$$

<그림 1> 페트리네트 예: PN_1

2. 관련 연구

본 논문에서 N은 음수가 아닌 정수를, R은 음수가 아닌 실수를, n는 페트리네트의 플레이스의 개수를($n = |P|$), m는 트랜지션의 개수를($m = |T|$), $e[j]$는 j번째만 1이고 나머지는 0인 m-벡터를 나타내기 위해서 사용된다.

페트리네트 (Petri Nets)

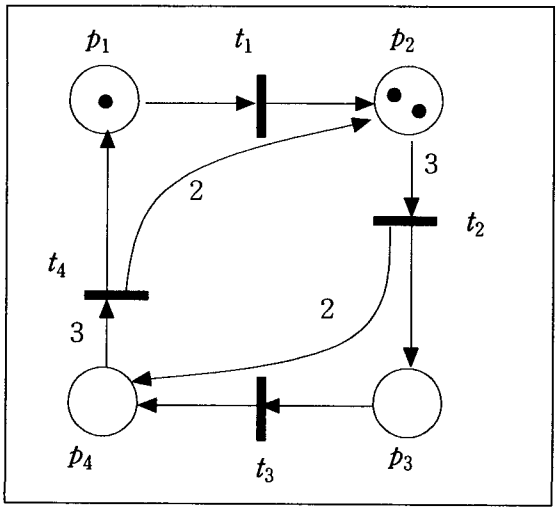
이 논문에서는 [3]에서 정의된 페트리네트 형식론에 약간의 수정을 한 것을 채용한다.

페트리네트는 다음의 구조를 갖는다.

$$PN = \langle P, T, I, O, \mu \rangle$$

P 플레이스(place)의 집합
T 트랜지션(transition)의 집합
$I: T \rightarrow P^\infty$ 입력함수
$O: T \rightarrow P^\infty$ 출력함수
$\mu: P \rightarrow N$ 마킹(marking)

입력함수 I와 출력함수 O는 트랜지션 T로 부터 장소의 가방(bag 혹은 multiset) P^∞로의 매핑(mapping)이다. I와 O는 필요에 따라서 두 개의 동등한 표현인 행렬 D^-와 D^+로 각각 사용



<그림 2> PN_1의 그림형태

될 수도 있다. 입출력 행렬과 입출력 함수는 다음과 같이 연관되어 진다.

$$D^-[j,i] = d_{ji}^- = \#(p_i, I(t_j))$$
$$D^+[j,i] = d_{ji}^+ = \#(p_i, O(t_j))$$

여기서 $d_{ji}^- = \#(p_i, I(t_j))$는 $I(t_j)$에 존재하는 p_i의 개수 혹은 p_i로부터 t_j로의 모서리(arc) 개수이고, $d_{ji}^+ = \#(p_i, O(t_j))$는 $O(t_j)$에 존재하는 p_i의 개수 혹은 t_j로부터 p_i로의 모서리 개수이다. 페트리네트의 행렬형태의 동등한 구조는 다음과 같다:

$$PN = \langle P, T, D^-, D^+, \mu \rangle$$

마킹(marking) $\mu: P \rightarrow N$은 플레이스로부터 음수가 아닌 정수로의 매핑이다. 마킹은 플레이스에 토큰(token)의 할당하는 것을 의미한다. 일반적으로 μ는 n-벡터로 표현된다.

$$\mu = (\mu_1, \mu_2, \cdots, \mu_n)$$

여기서, $n = |P|$이고, $\mu_i = \mu(p_i)$는 플레이스 p_i에 있는 토큰의 개수를 나타낸다.

페트리네트의 실행은 그 페트리네트가 가지는 토큰의 수와 분포에 의해 제어된다. 페트리네트는 트랜지션을 fire하여 실행된다. 트랜지션의 fire는 그 트랜지션의 입력 플레이스에서 토큰을 제거하고 출력 플레이스에 토큰을 추가하는 과정을 거친다. 임의의 트랜지션 t_j가 fire할 수 있으려면 t_j는 반드시 enable되어 있어야 한다. t_j는 다음의 조건을 만족할 때 enable되었다고 정의된다:

$$\mu(p_i) \geq \#(p_i, I(t_j)) \quad \text{for all } p_i \in P$$

행렬식으로 표현한 enable 조건은 다음과 같이 표현된다:

$$\mu \geq e[j] \cdot D^-$$

위의 조건을 만족하지 않으면 t_j는 disable되었다고 말한다.

마킹 μ에서 enable된 모든 트랜지션의 집합을 $E(\mu)$로 표현한다.

$$E(\mu) = \{ t_j \mid \mu \geq e[j] \cdot D^- \wedge 0 \leq j \leq m \}$$

페트리네트의 시뮬레이션 규칙은 다음과 같다:

(i) 현재의 마킹 μ에서 enable된 모든 트랜지션 $E(\mu)$을 찾는다. $E(\mu)$을 구하기 위해, 각 트랜지션 t_j에 대해서 다음을 테스트 하여야 한다.

$$\mu_i \geq d_{ji}^- \quad \text{for each } i = 1..n$$

ii) 무작위로 $E(\mu)$중의 한 트랜지션을 선택한다. 선택된 트랜지션을 t_j라고 가정하자.

(iii) 선택된 트랜지션 t_j을 fire한다. 새로운 마킹 μ'는 다음과 같이 유도된다:

$$\mu_i' = \mu_i - d_{ji}^- + d_{ji}^+ \quad \text{for each } i = 1..n$$

μ'이 μ로부터 유도된 사실을 $\mu \rightarrow \mu'$으로 표현하고 μ'은 μ로 도달할 수(reachable) 있다고 한다. 초기 마킹 μ로 부터 하나 혹은 여러 개의 트랜지션을 fire하여 도달할 수 있는(reachable) 모든 마킹 집합을 $R(\mu)$로 나타낸다.

어떤 페트리네트의 각 플레이스의 토큰이 유한한 수 k을 초과하지 않으면, 그 페트리네트는 k-한정(k-bounded)이라고 부른다.

<그림 1>은 예제 페트리네트 PN_1을 보여주고 있다. <그림 2>는 <그림 1>과 동등한 명세방법인 그림 형태로 표현된 PN_1을 보여주고 있다. 모서리 (u, v) 위의 정수 w는 그 모서리의 무게(weight) 즉 u에서 v로 w개의 병렬 모서리가 존재하는 것을 나타낸다. 예를 들면, (p_4, t_4)는 무게로 3를 가지는 것은 p_4에서 t_4로 세 개의 모서리가 있는 것을 나타낸다. 초기 마킹 (1,2,0,0)으로부터 도달될 수 있는 모든 마킹에 대해서, 각 플레이스의 토큰 개수가 3이하이므로 PN_1는 3-한정 페트리네트이다.

3. 데이터 중첩 방법

페트리네트의 시뮬레이션 규칙에 따르면, 시뮬레이션 동안 사용되는 연산자는 세 개 뿐이다. 즉 음수가 아닌 정수에 대한 세 개의 연산자 ($\geq$, -, +)만이 사용된다. 그러므로 만일 μ과 D^-와 D^+의 각 행을 적절히 잘 중첩시키면, 일반 프로세서의 SUB, ADD와 같은 일반적인 명령어로 SIMD-타입의 중첩 명령어 효과를 낼 수 있게 된다.

μ과 D^-와 D^+의 각 행을 적절히 잘 중첩시키기 위해서 우선 페트리네트가 한정되어야 한다. 만약 주어진 페트리네트가 k-한정이라면, μ로부터 도달되는 어떤 마킹에서도 각 플레이스의 토큰 값이 k를 초과할 수 없다. 즉 k는 상한 경계 값이다.

한편 D^-와 D^+도 시뮬레이션 중에 사용되므로, 두 행렬의 원소 값도 고려되어야 한다. $d_{\max}$가 D^-와 D^+의 원소 중 가장 큰 값을 나타내면, 시뮬레이션 도중에 계산에 사용되는 가장 큰 수는 다음과 같게 된다.

$$x = \max\{k, d_{\max}\}$$

즉 x는 페트리네트 시뮬레이션 중에 나올 수 있는 μ, D^-와 D^+의 원소 중 가장 큰 값이다. 주어진 k-한정 페트리네트에서 x가 알려지면, μ과 D^-와 D^+의 원소를 저장하기 위해 필요한 최소의 비트(bit)수는 $\lceil \log_2^{x+1} \rceil$가 된다. $|R|$는 사용하려는 프로세서의 레지스터의 비트수를 나타내기 위해 사용된다. 예를 들면, DEC Alpha 프로세서는 $|R|$=64가 된다.

따라서 하나의 레지스터에 다음 수식만큼의 원소들이 중첩될 수 있다.

$$\left\lfloor \frac{|R|}{\lceil \log_2^{x+1} \rceil} \right\rfloor$$

μ과 D^-와 D^+의 각 원소를 저장하기 위해 $\lceil \log_2^{x+1} \rceil$ 비트를 사용한 두 레지스터 값을 일반적인 +나 -를 적용하면 각각의 중첩된 데이터들 끼리 더해지거나 빼진다. 이유는 다음과 같다. +나 -연산자는 시뮬레이션 규칙의 세 번째 단계에서 새로운 마킹을 구하기 위해 항상 동시에 사용된다. 이 단계에서 사용되는 수식은 편리를 위해 다시 기술하면 다음과 같다. (t_j가 fire 한다고 가정)

$$\mu_i' = \mu_i - d_{ji}^- + d_{ji}^+ \qquad \text{for each } i = 1..n$$

μ_i'을 구하기 위해서 μ_i값에서 d_{ji}^-값을 뺀 후 d_{ji}^+을 더한다. 페트리네트 특성상 항상 다음의 조건을 만족한다. 즉 t_j의 enable 조건이 $0 \leq \mu_i - d_{ji}^-$이므로 다음의 수식이 만족된다.

$$0 \leq \mu_i - d_{ji}^- \leq x$$

또한 다음의 수식에서 μ_i'는 새로운 마킹에서 p_i에 존재하는 토큰의 개수이므로 x정의에 의해 다음의 조건이 만족된다.

$$0 \leq (\mu_i - d_{ji}^-) + d_{ji}^+ \leq x$$

따라서 시뮬레이션의 (iii)단계의 연산에서는 $\lceil \log_2^{x+1} \rceil$ 비트를 사용하여 데이터를 중첩시키고 일반 연산자(-,+)을 사용함으로써 SIMD-타입의 병렬효과를 얻을 수 있다.

하지만 시뮬레이션의 (i)단계에서 사용되는 비교연산자 $\geq$에 대응하는 중첩된 데이터에 대한

연산자 ⊐를 효율적으로 구현하기 위해 다음과 같이 중첩시킨다. 각 중첩된 요소를 저장하기 위해 $\lceil \log_2^{x+1} \rceil$ 비트 외에 일종의 부호비트 역할을 하는 1 비트를 추가하여 $\lceil \log_2^{x+1} \rceil + 1$ 비트를 사용한다. 이것에 대한 이유는 다음과 같다. ⊐를 구현하기 위해 중첩된 데이터에 대해서 뺄셈 연산자 -를 사용하고자 한다. 여러 개의 데이터가 중첩된 두개의 레지스터 값들 (left, right)이 있을 때, (left ⊐ right)가 참을 리턴하려면 left의 각 요소 값이 right의 각 요소 값들보다 크거나 같아야 한다. left의 각 요소 값이 right의 각 요소 값들보다 크거나 같을 경우 (left - right)는 산술적 underflow가 발생하지 않는다. 만약 underflow가 발생하면 left의 어떤 요소 값이 right의 대응하는 요소 값보다 적다는 것을 의미한다. (left - right)했을 때 요소 값에서 일어나는 산술적 underflow를 효율적으로 검출하기 위해, 각 요소 값을 저장하기 위한 $\lceil \log_2^{x+1} \rceil$ 비트 외에 부호 비트 역할을 하도록 각 요소 값의 MSB에 1비트를 추가한다. 추가된 이 비트들은 0로 초기화한다.

이와 같은 데이터 중첩 방법을 사용하면 (left ⊐ right)의 결과 값을 얻기 위해 (left - right) 값을 구한 후 각 요소에 추가된 비트의 값으로 결과를 판단할 수 있다. 즉, 각 요소에 추가된 비트의 값이 적어도 하나가 1이면 (left ⊐ right)이 거짓인 것을 나타내고 모두가 0이면 참을 나타낸다. 따라서 SIMD-타입의 병렬효과를 어느 정도 얻을 수 있다.

결론적으로, μ, D^-와 D^+의 각 원소를 저장하기 위해서 $\lceil \log_2^{x+1} \rceil + 1$ 비트를 사용하여야 한다.

최대 중첩율(MPF)는 다음과 같이 정의된다:

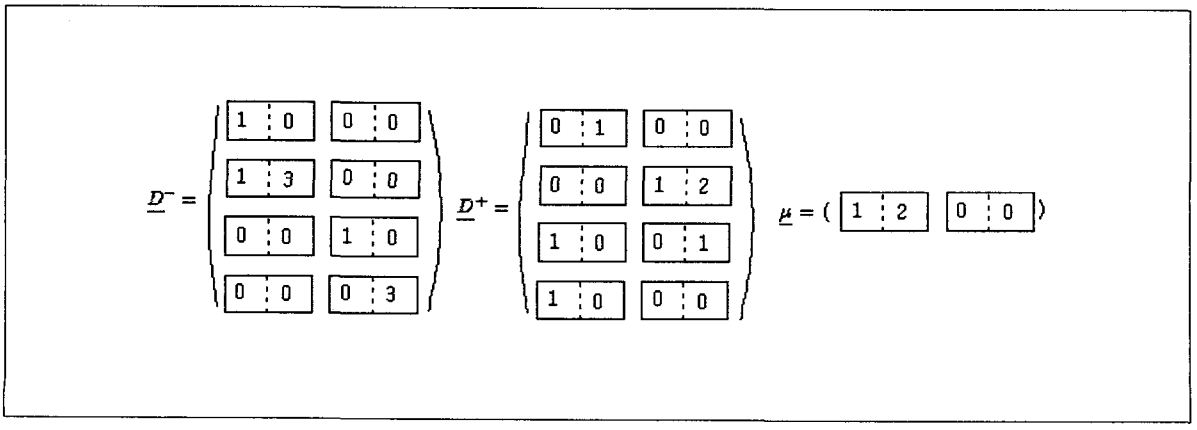
$$MPF = \left\lfloor \frac{|R|}{\lceil \log_2^{x+1} \rceil + 1} \right\rfloor$$

그러므로 실제 우리가 선택할 수 있는 중첩율(PF)는 다음과 같다.

$$1 \le PF \le MPF$$

만약 PF가 선택되면, 우리는 μ과 D^-와 D^+의 각 행의 연속되는 PF개의 원소들을 중첩시킬 수 있다. 중첩된 개념을 나타내는 기호는 위해서 비 중첩된 개념을 나타내는 기호에 밑줄을 사용할 것이다. 그러므로 μ과 D^-와 D^+에 대응되는 중첩된 데이터는 $\underline{\mu}$과 $\underline{D}^-$와 $\underline{D}^+$이 된다.

예로서 3-한정 페트리네트인 PN_1을 살펴보자. 만약 $|R|$=32라고 가정하면, PN_1의 MPF는 다음과 같다.



<그림 3> PN_1의 중첩된 데이터 (MPF=10, PF=2)

<표 1> SUN SPARCstation에서 GNU gcc 컴파일러를 사용한 어셈블리어 코드

논리적 연산자	GNU gcc 언어	어셈블리어

$l \geq r$	int l, r;
	l >= r
		ld [%fp-20],%o0
		ld [%fp-24],%o1
		cmp %o0,%o1
		bl .LL2

$l \sqsupseteq r$	int l, r;
	((l-r)&EXTRA_BITS)
		ld [%fp-20], %o0
		ld [%fp-20], %o1
		sub %o0,%o1,%o0
		sethi %hi(-2139062144),%o2
		or %o2,%lo(-2139062144),%o1
		and %o0,%o1,%o0
		cmp %o0,0
		be .LL3

$$MPF = \left\lfloor \frac{32}{\lceil \log_2^{3+1} \rceil + 1} \right\rfloor = 10$$

<그림 3>은 PF=2로 선택했을 때의 중첩된 데이터 μ과 $\underline{D}^-$와 $\underline{D}^+$를 보여주고 있다.

4. SIMD-타입의 병렬 시뮬레이션 알고리즘

2절에서 소개된 것처럼 페트리네트의 시뮬레이션 규칙은 다음의 3단계로 구성된다.

(i) enable된 트랜지션의 집합 $E(\mu)$구한다.

(ii) $E(\mu)$에서 fire할 트랜지션을 선택한다.

(iii) 선택된 트랜지션을 fire한다.

이 논문에서는 단계 (i)과 (iii)를 SIMD-타입의 병렬화를 시킨다.

임의의 PF로 중첩된 페트리네트 $\underline{PN}$를 고려하자.

$$\underline{PN} = \langle P, T, \underline{D}^-, \underline{D}^+, \mu \rangle$$

첫 번째 단계 병렬화

시뮬레이션 알고리즘의 첫 번째 단계는 다음과 같이 병렬화하기 위해 재 정의된다.

$$\underline{m}_i \sqsupseteq \underline{d}_{ji}^- \quad \text{for} \quad i = 1.. \left\lceil \frac{n}{PF} \right\rceil$$

여기서 $\sqsupseteq$는 왼쪽 피연산자 내에 중첩되어 있는 각각의 요소 값이 오른쪽 피연산자의 대응하는 요소 값보다 크거나 같을 경우에 참을 리턴하고, 아닌 경우에는 거짓을 리턴한다. 이 연산자를 C언어로 구현하면 다음과 같이 된다. $l \sqsupseteq r$이라고 가정하면, 앞 절에서 설명하였듯이 C언어로 다음과 같이 구현될 수 있다.

((l-r) & EXTRA_BITS) == 0

여기서 EXTRA_BITS은 길이가 $|R|$비트이고, 중첩된 각 요소의 $\lceil \log_2^{x+1} \rceil$ 비트들은 0이고 부수적으로 추가된 각 비트에 해당하는 위치는 1을 가진다. $l \sqsupseteq r$의 구현은 이런 사실을 이용한다. 즉 l의 중첩된 각 요소 값이 r의 중첩된 각 요소 값보다 크면 $l - r$의 결과값은 EXTRA_BITS의 1에 대응되는 비트들의 값으로 모두 0을 가진다. 반면에 적어도 하나의 l의 중첩된 요소 값이 r의 중첩된 요소 값보다 작으면, $l - r$의 결과값은 산술적 언더플로우 (underflow)가 일어나면서 EXTRA_BITS의 1에 대응되는 비트들 중 적어도 하나는 1을 가지게 된다. 따라서 l의 중첩된

<표 2> 1-한정 식사하는 철학자 문제의 실험 데이터

	DEC 워크스테이션 64비트 알파 프로세서 $\|R\|=64, MPF=32$						SUN SPARCstation 20 32비트 SPARC 프로세서 $\|R\|=32, MPF=6$				
알고리즘	old	new					old	new			
PF	1	2	4	8	16	32	1	2	4	8	16
T_{total}(초단위)	17.22	9.79	4.76	2.82	1.69	1.30	19.41	11.05	5.97	3.47	2.24
성능증가비 (old/new)		1.76	3.47	6.11	10.19	13.25		1.76	3.25	5.59	8.09
성능증가비/PF (%)		88.0	86.8	76.4	63.7	41.4		88.0	81.3	69.9	54.3

데이터 중에서 하나라도 r의 중첩된 데이터보다 적으면 위의 구현은 거짓을 리턴하고, 아니면 참을 리턴한다.

논리적 비교연산자 ⊐는 하나의 트랜지션이 enable되었는지 여부를 파악하기 위해 $\lceil \frac{n}{PF} \rceil$번 실행되어야 한다. 이후로는 $\lceil \frac{n}{PF} \rceil$를 나타내기 위해서 $\underline{n}$를 사용한다. ⊐의 실행시간 $ET(\sqsupset)$은 ≥의 실행시간 $ET(\geq)$보다 좀더 걸리기 때문에 첫 번째 시뮬레이션 단계에서의 수행시간증가율은 다음과 같이 정의된다.

$$\frac{n \cdot ET(\geq)}{\underline{n} \cdot ET(\sqsupset)} \approx PF \cdot \frac{ET(\geq)}{ET(\sqsupset)}$$

여기서 $\frac{ET(\geq)}{ET(\sqsupset)}$는 사용하는 프로세서에 대응하는 상수값이다. 예를 들면, <표 1>에서 보여지듯이, 썬 SPARCstation 20 컴퓨터의 GNU gcc 언어를 사용하여 비교하면, $ET(\geq)$는 4 명령어를 $ET(\sqsupset)$는 8 명령어로 구성된다.

세 번째 단계 병렬화

세 번째 단계의 병렬화는 다음과 같다. 중첩된 마킹 $\underline{\mu}$에서, 트랜지션 t_j가 fire할 때 새로운 마킹 $\underline{\mu}'$는 다음과 같이 정의할 수 있다.

$$\underline{\mu}' = (\underline{\mu}_i \ominus \underline{d}_{ji}^{-}) \oplus \underline{d}_{ji}^{+} \quad \text{for } i = 1..\underline{n}$$

여기서 ⊕과 ⊖ 중첩된 데이터를 각각 더하고 빼는 연산자이다. 앞 절에서 설명한 것처럼 한정 페트리네트의 경우에는 ⊕과 ⊖는 양 피연산자가 보통의 데이터인 것으로 간주하고 일반적인 +와 -연산으로 대치할 수 있다. 그러므로

$$\underline{\mu}' = (\underline{\mu}_i - \underline{d}_{ji}^{-}) + \underline{d}_{ji}^{+} \quad \text{for } i = 1..\underline{n}$$

따라서, 이 단계에서의 병렬화를 통한 성능향상은 다음과 같다.

$$\frac{n \cdot (ET(-) + ET(+))}{\underline{n} \cdot (ET(-) + ET(-))} \approx PF$$

5. 실험

저자들이 제안한 병렬알고리즘의 효율성을 확인하기 위해 다음의 두 개의 환경에서 실험을 하였다:

(i) 64 비트 알파 프로세서를 사용하는 DEC-3000 워크스테이션 (DEC OSF/1 OS)

(ii) 32 비트 SPARC 프로세서를 사용하는 SPARCstation 20 워크스테이션 (Solaris 2.4 OS)

실제 성능향상 및 PF(이론적 상한선)사이의 상관관계에 초점을 맞추기 위해 다음의 두 페트리네트를 사용하였다:

(i) 1-한정 식사하는 철학자 페트리네트 (dining philosophers problem)

(ii) 42개의 읽는(read) 프로세스와 한개의 쓰는(write) 프로세스로 구성된 42-한정 읽기-쓰기 페트리네트(readers/writers problem)

<표 3> 42-한정 읽기/쓰기 문제의 실험 데이터

	DEC 워크스테이션 64비트 알파 프로세서 $\|R\|=64, MPF=32$				SUN SPARCstation 20 32비트 SPARC 프로세서 $\|R\|=32, MPF=6$		
알고리즘	old	new			old	new	
PF	1	2	4	8	1	2	4
T_{total}(초단위)	22.12	12.87	6.60	3.68	27.16	15.58	8.14
성능증가비 (old/new)		1.72	3.35	6.01		1.74	3.33
성능증가비/PF (%)		86.0	83.8	75.1		87.0	83.3

기존의 알고리즘에 대한 제안된 알고리즘의 성능 향상 비는 PF를 2에서 MPF까지 변화시키면서 측정되었다. 단 PF는 2의 지수 승으로 한정하였다. 결과는 <표 2>와 <표 3>에 보여 지고 있다.

두 표에서 T_{total}는 40,000번의 트랜지션 fire동안 걸린 총 시뮬레이션 시간이다. 각 표의 old열은 기존의 시뮬레이션 알고리즘에 대한 실험 데이터이고 new열은 제안된 병렬 시뮬레이션 알고리즘에 대한 실험 데이터이다.

두 표에서 볼 수 있듯이 제안된 알고리즘의 성능향상은 선형은 아니지만 PF에 비례하여 증가함을 볼 수 있다. 특히 2-한정 페트리네트인 경우 DEC 워크스테이션에서 최고 13.25배나 빠르게 실행됨을 보여주고 있다.

6. 결론

한정(bounded) 페트리네트를 효율적으로 시뮬레이션할 수 있는 SIMD-타입의 병렬시뮬레이션 알고리즘을 제시하였다. 이 병렬알고리즘은 하나의 레지스트에 여러 데이터를 중첩한 데이터를 처리하는 방식이다. 제안된 알고리즘의 효율성을 보여주기 위해서 32비트와 64비트인 2개의 워크스테이션 환경에서 실험을 하였고 PF을 증가시킴에 따라 "성능증가비/PF"이 포화되어 가지만 "성능증가비"은 비례하여 증가하는 것을 보여주고 있다. 특히, 2-한정 페트리네트를 PF=32로 했을 때 최고 13.15배나 빨리 시뮬레이션 속도가 증가함을 보여 주었다.

참고문헌

[1] Intel Corporation, *The Complete Guide to MMX Technology*, McGraw-Hill, 1997.

[2] SunSoft, *SPARC Assembly Language Reference Manual*

[3] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceeding of the IEEE*, Vol 77, No. 4, pp. 541-580, Apr. 1989.

[4] J. L. Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[5] G. Chiola and A. Ferscha, "Distributed Simulation of Petri nets," *IEEE Parallel & Distributed Technology: Systems and Applications*, Vol. 1, No. 3, pp 33-50, 1993.

[6] F. Baccelli, N. Furmento and B. Gaujal, "Parallel and Distributed Simulation of Free Choice Petri Nets," *Proceedings of Parallel and Distributed Simulation 1995*, pp 3-10.

[7] D. M. Nicol and W. Z. Mao, "Automated Parallelization of Timed Petri-Net Simulations," *Journal of Parallel and Distributed Computing*, Vol. 29, No. 1, Aug 1995, pp. 60-74

저자소개

김영찬

1985: 아주대학교 공과대학 전자공학과 학사
1987: 한국과학기술원 전기및전자공학과 석사
1995: 한국과학기술원 전기및전자공학과 박사
1984 - 1990: 삼성전자 종합연구소 연구원
1995 - 1995: 한국과학기술원 위촉연구원
1997 - 1998: 시스템공학연구소 및 전자통신연구원 연구원
1998 - 현재: 한밭대학교 정보통신-컴퓨터공학부 조교수
관심분야: 시스템 검증, 형식론(Petri Net, DEVS등), 데이터베이스, XML

김탁곤

1988: 아리조나대학교 전자/컴퓨터공학과 공학박사
1980 - 1983: 국립수산대학(현: 부경대학교) 통신공학과 전임강사
1987 - 1989: 아리조나 환경연구소 연구 엔지니어
1989 - 1991: 캔사스 대학교, 전자전산학과, 조교수
1991 - 현재: 한국과학기술원, 전자전산학과, 조교수/부교수/교수
2001 - 현재: 미국 시뮬레이션 기술사
2000 - 현재: SIMULATION: Trans of SCS 편집위원장
모델링/시뮬레이션 방법론 및 환경 개발, 시뮬레이션에 기반한 시스템 분석 등에 관하여 국제적으로 활발한 연구 활동을 하고 있으며 국제학술지/국제학술발표대회에 100 편 이상의 논문을 개재 하였음. 시뮬레이션 모델링에 관련된 다수의 국제학술지 편집을 담당하고 있다. 저서(공저자: B.P. Zeigler, H. Praehofer)로 2000 년 미국 Academic Press에서 발간한 모델링 시뮬레이션 전문 교재인 *Theory of Modeling and Simulation(2nd edition)* 이 있음. 한국시뮬레이션 학회 초대 편집위원장을 역임하였고 현재 학술부 회장 임. IEEE(국제 전기전자 학회) 및 SCS(국제 시뮬레이션 학회)의 Senior Member 이며 ACM(미국전산학회) 및 Eta Kappa Nu 의 Regular Member 임.