Journal of the Korea Academia-Industrial cooperation Society
Vol. 14, No. 2 pp. 858-867, 2013

http://dx.doi.org/10.5762/KAIS.2013.14.2.858

Petri Net 이론 관점에서 본 소프트웨어 혁신의 확산

한지연[1], 안종창[1*], 이욱[1]
[1]한양대학교 정보시스템학과

Diffusion of software innovation: a Petri Net theory perspective

Jiyeon Han[1], Jongchang Ahn[1*] and Ook Lee[1]
[1]**Department of Information System, Hanyang University**

요 약 본 연구에서는 MPSOC(Multiprocessor System-on-Chip) 환경의 소프트웨어적인 측면을 주 연구대상으로 하였고 범용 전문 프로그래머들에게 병렬 애플리케이션 프로그래밍을 위한 패턴언어를 제공한다면 병렬처리의 장점을 호소할 수 있을 것으로 보았다. 특히 자체적인 분류에 따른 Data, Tasks, Data flow 모델을 선별하고 그중 CUDA와 HOPES를 경험한 대상으로 Petri Net을 활용, 그들의 숙련도를 확인하는 과정을 포함한다. 각 영역의 숙련 정도, 서로 다른 모델에 대한 이해도를 실험을 통해 검증하였다. 페트리넷은 병렬프로그래밍의 설명에 용이한 모델로 특히 동시성과 병렬성을 설명하는데 탁월한 모형을 제시 할 수 있었다. 실험대상자들에게 페트리넷에 관한 4시간의 선행학습을 시킨 후 56명에게 실험을 실시하여 독립표본 t-검정을 수행했다. 비록 설정된 두 가지 가설이 지지되지 않았지만, 각 영역에서의 숙련자들이 Tasks 중심 혹은 Data 중심의 모형을 얼마나 상호 이해하는가를 판단 할 수 있었다.

Abstract Hardware and software field are developed by environment of MPSOC. Also it is still working with economic world and academic world. This study focus on software side and try to classify from parallel programming design world. It can be divided by three; Data, Tasks, and Data flow model. Then we used Petri Net to CUDA and HOPES programmer and found how much they understand parallel programming for each side. We focus on two sides and what is different between their experience. Petri Net is easy to descript parallel program or parallel design pattern for Task, Data, and Hybird. This research can explain how they know and how much they know about parallel programming.

Key Words : Petri Net, Parallel programming, Data model, Task Model, Hypothesis testing

1. 서론

1.1 연구의 목적

지난 수십 년 동안 사용되어 온 병렬 하드웨어는 이제 주류가 되어 가고 있다. 반면, 이런 종류의 소프트웨어는 만들기가 너무 힘들다는 고정관념 때문에 병렬 하드웨어의 장점을 취한 소프트웨어는 다소 희귀하다[1, 2]. 병렬 하드웨어에 대한 대부분의 연구는 병렬 하드웨어의 사용을 더욱 쉽게 만들고 특정한 컴퓨터 시스템의 상세사항을 프로그래머로부터 숨겨서 이동형 소프트웨어를 만들 수 있도록 하는 병렬 프로그래밍 환경의 형성에 초점을 맞춰 왔다[1]. 이런 종류의 프로그래밍 환경의 좋은 실례로서 메시지 전달 인터페이스(MPI; Message Passing Interface)를 들 수 있다[6]. 이 환경은 고성능 처리(HPC; high-performance computing) 커뮤니티의 니즈를 전반적으로 충족시켜 왔다. 그러나 HPC 커뮤니티 밖에서는 아주 일부의 프로그래머들만이 병렬 소프트웨어를 만드는 것을 고려해 본다는 점에서, 병렬 프로그래밍 환경을 연구하는 연구자들은 일반적인 성공을 거두었다고 볼 수 없다. 프로그래밍 분야의 나머지가 부담하는 비용으로

*Corresponding Author : Jongchang Ahn (Hanyang University)
Tel: +82-10-9127-2595 email: ajchang@hanyang.ac.kr
Received November 7, 2012 Revised November 28, 2012 Accepted February 6, 2013

HPC 커뮤니티에만 초점을 맞추는 연구를 하는 것으로 볼 수 있다.

이런 현상이 일어난 이유는 복잡하다. 병렬 프로그래밍 환경에 오류가 존재하지만 병렬 컴퓨터의 높은 수준의 추상성으로는 고칠 수 없다고 믿어왔다. 범용 전문 프로그래머가 병렬 소프트웨어를 만들도록 하려면 그들이 알고리즘에서의 동시성을 이해하고 표현할 수 있도록 도와주는 프로그래밍 환경을 제공해야 한다. 이를 위해 우선, 병렬 프로그램은 디버깅이 어렵기 때문에, 혹은 하드웨어의 불일치 등의 난관에서 프로그래머들이 버그를 피할 수 있도록 도와주어야하며, 범용 전문 프로그래머들이 접근할 수 있는 방식으로 제시될 수 있어야 한다.

프로그래머들이 고품질의 소프트웨어를 설계하도록 돕는 것은 소프트웨어 공학의 일반적인 주제이다. 현재 가장 인기 있는 솔루션 중 하나는 설계패턴에 기반을 둔 것이나 디자인패턴에 기반을 둔 것이다. 설계 패턴은 소프트웨어 설계상 반복되는 문제에 대응하기 위해 세심하게 만들어진 솔루션이다. 이 패턴들은 일반적으로 계층을 조직한다. 소프트웨어 설계자들은 설계를 완료하기 위해 문제를 구체화하는 데서 이 패턴을 활용하여 스스로 지침을 삼을 수 있다. 이러한 패턴들을 총괄하여 패턴언어라고 한다. 효율적인 패턴 언어는 패턴 자체를 넘어서는 것이며, 소프트웨어 설계자들에게 패턴을 사용하는 방법에 대한 지침을 준다.

설계패턴은 근래의 소프트웨어 설계에 주요한 영향을 미쳐 왔다. 본 연구는 프로그래머들이 병렬 소프트웨어를 만드는 것을 돕기 위해 위와 동일한 접근법을 활용할 수 있을 것으로 본다. 즉, 범용 전문 프로그래머들에게 병렬 애플리케이션 프로그래밍을 위한 패턴언어를 제공한다면 병렬처리의 매력을 호소할 수 있을 것이다.

이에 본 연구는 현재 사용되고 있는 다양한 병렬 컴퓨팅의 기법을 패턴언어와 관계된 뼈대, 프레임워크, 원형을 통해 정리할 것이고 이를 통해 상위 다른 디자인패턴을 사용하는 사용자들을 통해 어떻게 병렬프로그래밍 기법이 나아가야 할 것인지를 찾아볼 것이다.

1.2 연구 방법

본 연구는 서로 다른 병렬프로그래밍의 디자인패턴을 분류하고 이론적 문헌연구[3-5]를 바탕으로 실증적 연구를 실시하였다. 서로 다른 디자인패턴을 공부한 프로그래머를 대상으로 Petri Net(이하 페트리넷)을 이용한 설문지를 통해 그들의 숙달 정도와 차이점을 알아보고 실증분석을 실시하였다. 설문의 답변을 분석하기 위해 SPSS 17.0을 사용하여 독립표본 t-검정을 하였다.

2. 이론적 배경

2.1 병렬 프로그래밍 디자인 패턴

소프트웨어 개발을 촉진시키기 위해 프로그램의 전체적인 구조에서부터 상세한 설계에 이르기까지 패턴을 규명하고 탐색하는 연구들이 상당히 많이 수행됐다. 이러한 연구들의 공통적인 주제는 프로그램의 효율적인 설계 및 실행의 특성을 가지고 있는 패턴을 탐색하고 이러한 설계 및 실행을 다른 수많은 애플리케이션에 다시 사용하기 위한 것이었다.

Cole[8]에서 최초로 언급된 알고리즘 뼈대는 매우 높은 레벨의 패턴을 보유하고 있다. 이들은 애플리케이션에 전체적인 구조를 부여하는 고차함수로 가정된다. 위의 애플리케이션은 애플리케이션에 특이적인 낮은 수준의 코드를 공급한다. 비록 프로그램의 뼈대 실행에 대해 연구가 수행되었지만, 설계의 재사용에 강조를 둔다. 프로그램의 프레임워크 [9]도 위와 마찬가지로 프로그램의 전체적인 조직을 구성하나, 좀 더 상세(detail)하고 도메인(domain)에 특이적인 경향이 있다. 낮은 수준의 함수를 제공하며 설계는 물론 코드의 재사용에 강조를 둔다.

현재 존재하는 설계패턴들은 꽤 낮은 수준의 설계 문제를 해결할 수 있기는 하나, 설계 패턴은 프로그램의 뼈대 및 프레임워크와 상반되게 다양한 레벨의 설계상 문제를 해결할 수 있다. 이들은 코드 재사용보다는 설계 재사용을 강조한다. 일반적으로 이들은 자신의 애플리케이션에서 조언과 함께 패턴에 관한 설명만을 제공한다. 설계 패턴 접근법의 가장 유용한 특성 중 하나는 각 패턴이 하나의 명칭을 갖고 있기 때문에 프로그램 설계자와 개발자가 공통된 어휘를 사용할 수 있다는 것이다. 패턴에 관한 초기 연구[10]에서는 대부분 오브젝트 주도형의 순차 프로그래밍을 다루었으나, 최근의 연구[4, 5]에서는 대부분이 매우 낮은 수준이기는 하지만 병렬 프로그래밍에 접근하고 있다. 이러한 연구의 목적은 패턴을 덜 일반적인 패턴 언어에 편입시키고자 하는 것이다.

프로그래밍 원형은 전술한 범주의 모든 요소를 조합시킨다. 이들은 높은(high) 레벨의 일반적인 처리 요소와 구조적 요소를 가진다. 그러나 높은 수준의 프레임워크와 낮은 수준의 코드의 라이브러리를 함께 실행할 수 있는 기초도 제공한다. 병렬 프로그래밍의 원형은 일반적인 처리 패턴을 병렬화 전략과 조합시킨다. 이렇게 조합된 패턴은 설계 패턴의 기능과 유사하게 프로그램의 설계와 추론에 대한 기초를 제공하는 한편, 프레임워크의 기능과 유사하게 코드의 뼈대와 라이브러리를 위한 기초도 제공한다. 그러나 프로그래밍 원형은 특정 문제에 적합한 원

형을 어떻게 선택하느냐에 대한 질문은 직접 해결하지 못한다.

설계 패턴은 하나의 콘텍스트(context) 내의 하나의 문제에 대한 솔루션이다. 본 연구는 관련 문제에 대한 높은 질의 솔루션을 면밀히 살펴봄으로써 설계 패턴을 발견했다. 설계 패턴은 시스템적인 방식으로 기록되고 저급의 솔루션으로부터 좋은 솔루션을 구분 짓는 공통 요소를 가지고 있다.

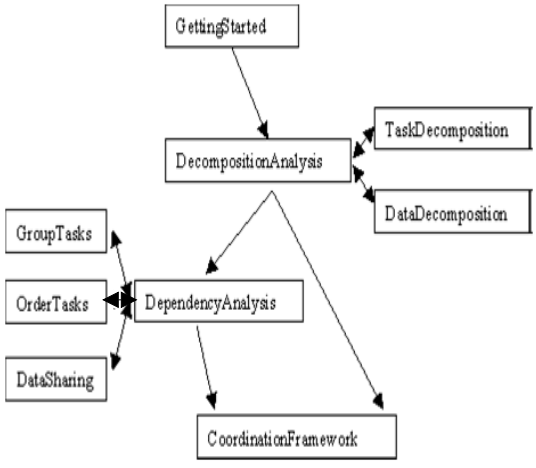
본 연구는 패턴의 측면에서 복잡한 시스템을 설계할 수 있다. 결정이 필요한 각 지점마다 설계자는 패턴 카탈로그로부터 적절한 패턴을 선택한다. 각 패턴은 다른 패턴으로 연결되어 최종적으로 패턴의 그물과 같은 설계가 된다. 이러한 설계 방식을 지원하는 패턴들의 카탈로그를 패턴 언어라고 부른다. 패턴 언어는 단순히 패턴들이 모인 한 개의 카탈로그를 의미하지 않는다. 패턴 언어에는 설계 방법론이 포함되어 있고 애플리케이션 설계자에게 도메인에 특이적인 조언(advice)을 제공한다. 용어의 중복에도 불구하고 본 연구의 패턴 언어는 프로그래밍 언어가 아니라 언어와는 무관한 패턴의 구조적 총괄이라고 할 수 있다.

여기서 기술하는 패턴 언어는 병렬 처리를 실행하기 위한 새로운 프로그램을 설계하고자 하는 애플리케이션 프로그래머를 위한 것이다. 최고 수준의 패턴은 동시 실행되는 구성요소들의 문제점을 어떻게 분해(decomposition)할 것인지를 선택할 수 있도록 돕는다. 낮은 수준의 패턴은 병렬 환경의 원시시대라고 할 수 있는 가장 낮은 수준에서 병렬 알고리즘의 관점에서 설계자들이 이러한 병렬제어를 표현할 수 있도록 돕는다. 이러한 낮은 수준의 패턴 언어는 일반적으로 패턴이라고 간주되지 않는 객체이다. 그러나 본 연구는 이러한 낮은 수준의 패턴을 문제 구체화로부터 코드까지의 완벽한 경로를 제공할 수 있도록 패턴 언어에 포함시켰으며, 프로그래머를 위한 일정한 인터페이스를 제공하기 위해 패턴 표기법에 사용하였다.

본 연구의 패턴 언어는 병행 제어만 해결한다. 즉 패턴 기반의 다른 설계 시스템을 대체하기 보다는 보완하고자 하는 것이다. 프로그래머들이 이 패턴 언어로 작업하기 위해서는 코드 오브젝트, 수학, 자신의 문제에 관계된 추상 알고리즘을 이해해야 한다. 이 패턴 언어에서, 본 연구는 문제 도메인 내와 패턴 언어 밖에서 일어나는 설계 활동을 “문제 공간에서의 추론”이라고 한다. 패턴 언어는 문제 공간에서 추론된 결과를 활용하게 되나, 프로그래머가 추론하는 것을 지원하지는 않는다.

본 연구는 패턴 언어의 구조를 알아보기 전에 먼저 한 가지 사항을 알아보아야 한다. 코드와 설계의 재사용을 촉진함에 더하여, 패턴은 일종의 “재사용의 검증”이 가능하도록 한다. 패턴은 자신의 애플리케이션에 특이적인 부분을 세심하게 구체화한다. 즉, 패턴에 대해 한번 수행되었던 검증으로부터 장점만을 사용하여 시정한다고 알려져 있다. 또한 실행을 포함하는 패턴에는 옳은 것으로 검증된 부분도 포함될 수 있다. 그러므로 패턴은 이론 과학자들의 커뮤니티로부터 프로그래머가 유용하게 활용할 수 있는 결과들을 전달해 주는 운반체로서 작용할 수 있을 것이다. 본 연구의 패턴 언어는 위와 같은 장점을 가지고 있다

패턴 언어의 구조는 한 개의 직선형 계층 내에 배열된 네 개의 코어 설계 공간 주위에 조직되어 있다. 높은 수준의 공간은 특정한 목표 환경과는 무관한 병렬 프로그래밍의 추상적인 관점에 해당한다. 낮은 수준의 공간은 실행에 좀 더 특이적이며, 특정한 목표 환경 내에서 병렬 프로그램을 구축하기 위해 낮은 수준의 벽돌이라는 패턴 기반의 표현을 사용한다. 본 연구의 패턴 언어를 활용하기 위해, 애플리케이션 설계자들은 최고 레벨에서 시작하여 점차 아래로 내려오면서 문제 표현을 병렬 알고리즘으로, 종국적으로는 병렬 코드로 전환시켜야 할 것이다. 동시성을 고려한 디자인 패턴으로 이 공간은 이용 가능한 병행을 노출시키는 것과 관련된다. 이 공간의 패턴은 프로그래머들이 동시 수행할 수 있는 작업단위의 문제들을 분해할 수 있도록 도와준다. 그림1은 이 설계 공간에서의 패턴과 알고리즘 설계자와 프로그래머들이 어떻게 사용할 수 있을지를 나타내고 있다. 이 그림은 병행검색 설계 공간을 나타내고 있다. 설계자들이 소프트웨어 설계 시에 이용 가능한 병행을 규정할 때 맨 위부터 시작하여, 화살표가 가리키는 방향대로 패턴을 따라 작업할 수 있다.

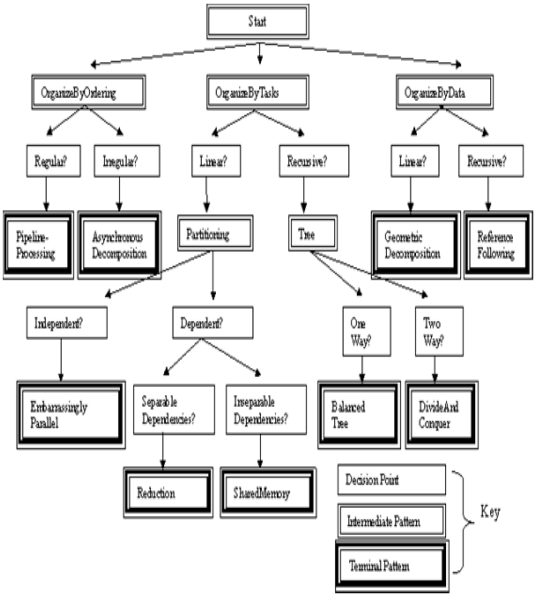


[Fig. 1] Design space for parallel searching

이 공간을 통한 경로는 개시패턴으로 시작한다. 이 패턴은 설계자가 처한 문제에 대한 정보를 어떻게 조직화할 것인가를 말해 준다. 다음 단계는 그 문제를 상대적으로 독립적인 단위로 분해하는 것이다. 이 분석법에는 두 부분이 존재하는데, 과업(task)에 의한 분석과 자료(data)에 의한 분석이다. 그림에서 이중 화살표가 가리키는 것은 설계자가 최종 분해를 발견하기 전에 두 개의 분해 간에 반복이 필요할 수 있다는 의미이다.

문제가 분해되고 나면 설계자는 각각의 과업이 서로 어떻게 의존하고 있는지를 결정해야 한다. 이 결정은 세 개의 패턴 간의 반복을 통해 실행된다. 'GroupTasks'패턴은 동시에 수행되어야 할 요소들을 집단으로 모을 수 있도록 돕는다. 'OrderTasks'패턴은 자료 또는 일시적인 의존도에 따라 태스크 또는 태스크 집단을 명령할 수 있도록 돕는다. 종국적으로 'DataSharing'패턴은 자료를 태스크 간에 어떻게 공유할 것인가를 규명하도록 한다.

알고리즘 구조의 디자인 패턴에서 이 설계 공간은 병행을 가능하게 하기 위한 병렬 알고리즘의 높은 수준의 구조와 전체적인 전략에 관한 공간이다. 이 공간에서 작업하는 설계자는 병행검색설계 공간에서 노출된 병행을 어떻게 사용할 것인지를 추론하게 된다. 이 공간의 패턴은 그림2에서 볼 수 있듯이 의사결정나무로 조직되어 있어, 설계자들을 자신의 문제에 적용될 패턴으로 이끈다.



[Fig. 2] Classification of final pattern

그림2의 최종 패턴이 알고리즘 설계에 사용된다. 이 그림의 의사결정나무는 특정한 최종 패턴에 도달하기 위해 설계자들이 사용하는 추론 과정을 보여주고 세 종류의 상자를 가지고 있다. 중간패턴, 의사결정지점, 최종패턴이 그것이다. 중간패턴은 설계자들이 설계 공간을 탐색할 수 있도록 하고, 의사결정지점은 중간패턴 내의 병행에 관해 만들어진 의사결정을 표현하며, 최종패턴이 알고리즘 설계에 사용된다.

설계자들은 의사결정나무의 가장 위에서 시작하여 가장 높은 수준의 조직을 가장 먼저 선택하게 된다. 이 지점에는 세 개의 선택이 존재하며, 각자의 선택은 의사결정나무의 주요 가지를 구성한다. 병행은 태스크 그룹의 일시적 명령, 문제의 태스크 분해, 또는 문제의 자료 분해의 관점에서 조직될 수 있다.

의사결정나무의 첫 번째 메인 가지는 “명령으로 조직” 패턴의 이하 부분으로서, 다시 2개의 가지로 나뉜다. 이들은 명령을 조직하는 방법을 선택하도록 한다. 이 중 한 가지 선택은 “규칙적” 명령으로서 알고리즘 수행 중 변경되지 않고, 다른 한 가지 선택은 “불규칙적” 명령으로서 규칙적 명령보다 역동적이고 예측하기가 힘들다. 이들 두 선택은 각자의 최종패턴으로 이어진다.

두 번째 메인 가지인 ‘OrderByTasks’ 패턴에는 태스크 기반 패턴이 포함되며 의사결정나무에서 가장 복잡한 가지이다. 설계자는 태스크를 선형 수집할 지, 재귀적 자료로 구축할 지에 따라 선택할 수 있다. 첫 번째 잔가지는 ‘Partitioning’ 패턴이라 부르며 다시 2개의 가지로 나뉜다. 이 중 하나는 태스크를 독립적으로 수행하는 ‘Embarrassingly Parallel’ 패턴이며, 다른 하나는 태스크 간에 의존하는 패턴이다. 후자의 패턴은 병행으로부터 의존도를 이동시킬 수 있으면 ‘Separable case’, 태스크가 병행 내에서 다루어져야 하는 ‘Inseparable case’로 나뉜다. Fig. 2의 의사결정나무로 되돌아가면 재귀적 태스크 구조에 해당하는 두 개의 케이스를 볼 수 있다.

세 번째 메인 가지인 “자료에 의한 명령” 패턴은 자료 분해의 관점에서 가장 잘 알려진 패턴이다. 여기에는 선형과 재귀적 자료 구조에 해당하는 두 개의 케이스가 있다.

하나 이상의 최종패턴에 도달할 때까지 이 나무를 사용해 작업하면 설계자들은 자신의 문제에서 병행을 가능하게 하기 위해 병렬 알고리즘을 구축하는 방법을 결정할 수 있을 것이다.

구조에 따른 디자인 패턴에서 이 설계공간은 알고리즘 구조 공간에서 패턴을 실행시키기 위해 사용했던 낮은 수준의 구성체에 관한 것이다. 이들은 SPMD (single program, multiple data)와 같은 프로그램 구성체를 대표하며, 공유대기행렬과 같이 널리 사용되는 공유 자료구조를 대표한다. 본 연구는 이 설계공간의 많은 요소들을 패턴으로서 생각하지 않으며, 공유대기행렬과 같은 일부는

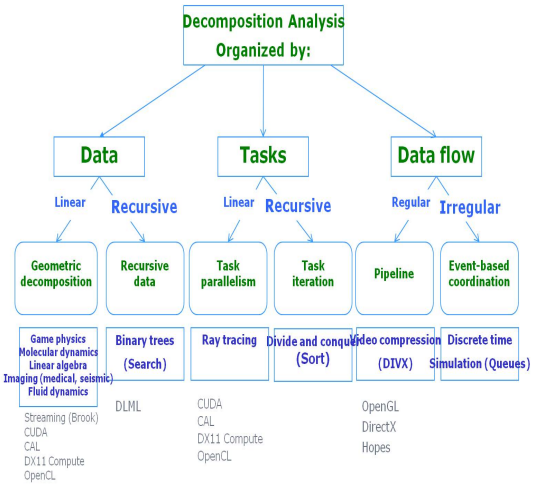
재활용 가능한 소프트웨어의 구성요소의 일부분으로서 프로그래머들에게 이상적으로 제공할 수 있을 것이다. 그럼에도 불구하고 본 연구는 이들을 패턴으로 기록했다. 그 이유는 첫째, 전술한 바와 같이 본 연구의 패턴언어의 모든 요소를 패턴으로 기록함으로써 프로그래머들에게 통일된 표기법을 제공할 수 있기 때문이다. 둘째, 이러한 패턴을 설명함으로써 자신의 실행을 형성하고자 하는 프로그래머들에게 지침이 될 수 있기 때문이다.

전술한 바와 같이 본 연구는 패턴언어로부터 얻은 경험으로 이 설계공간의 내용이 변화할 것으로 기대한다. 특히, 공유 자료구조 목록은 불완전하다. 결국 모든 주요한 공유 자료구조를 포함시키도록 확장되어야 할 것이다.

실행에 의한 디자인 패턴에서 이 설계공간은 태스크가 자신의 행동을 조율하는 방법에 관한 것이다. 프로세스/스레드 관리를 위한 공통 기전을 패턴 기반으로 설명하기 위해 이 공간을 활용한다. 즉, 프로세스/스레드의 생성과 삭제, 프로세스/스레드의 상호작용, 모니터 또는 메시지 전달 등이다. 이 설계공간의 구성요소들은 대개 높은 수준의 설계 공간으로부터 패턴을 실행하기 위해 사용된다. 다른 설계공간은 수준이 높고 프로그래머들에게 특수한 프로그래밍 환경을 강요하지 않으나 이 공간은 설계의 실행을 위해 프로그래머들이 사용한 근본적인 기전에 더욱 근접한다. 본 연구는 문제 구체화부터 코드까지 완벽한 경로를 제공하고 표기법의 일치를 위해 이 패턴을 본 연구의 패턴언어에 포함시켰다.

2.2 패턴분류에 따른 병렬컴퓨팅 기법

위에서 제시한 최종 패턴 구조를 바탕으로 현존하는 다양한 병렬 컴퓨팅의 분류는 그림 3과 같다.



[Fig. 3] Computing classification by pattern

현재 사용되고 있는 디자인패턴은 어느 하나의 종류만을 사용하고 있는 패턴도 있지만 대부분의 구조는 다양한 방법을 차용하고 복합적으로 사용하고 있는 구조이다. 본 연구에서는 이중 CUDA와 HOPES모델의 숙련자를 실험군으로 하였다.

이상 연구 설계 및 실증분석에서의 대상군 선정에 있어 HOPES와 CUDA를 실험대상군으로 잡은 이유를 설명하기 위해 그림1부터 3까지의 패턴모델의 차이점을 설명하였다. 그림1은 설계단에서 프로그래머 혹은 설계자들이 동시성을 제어할 때 어떠한 의사결정을 하게 되는지를 설명하는(설계자, 개발자의 경험중심의 사고 혹은 개발능력에 비례하여) 것이다. 그림2는 그림1의 기법을 각각의 단계에서 집합화, 세분화(그 방법에 대해서는 그림2의 화살표와 함께 있는 물음표 질문을 이용하여) 디자인 패턴으로 분류한 모델이다. 그림3은 그림1과 그림2를 통해 구분지어진 디자인 패턴을 현존하는 실질 디자인 기법으로 분류해 놓은 것이다. 이에 따라 본 연구는 태스크 중심으로 동시성 제어를 병렬화한 CUDA와 Data flow 중심이며 파이프라인구조로 디자인한 HOPES를 비교 실험군으로 선정하게 되었다.

3. 연구모형 및 가설 설정

3.1 Petri Net

페트리넷은 비동기적인 요소들이 서로 긴밀하게 연결되어 있는 시스템을 모델링하기 위해서 제안된 방법이다. 페트리넷에서 시스템 구성 요소들 사이의 연결관계는 시스템에서 발생하는 이벤트의 인과 관계를 나타낸다. 기본적으로 장소의 집합 P, 전이의 집합 T, 입력 함수 I, 출력 함수 O의 4개의 집합으로 구성되어 있다. 입력함수 I는 전이 t1과 장소의 집합 사이의 관계를 나타내며, I(t1)로 표현된다. 출력함수 O는 전이 t1과 전이의 출력값의 집합 O(t1)로 표현된다[7].

- 정의 : 다음과 같은 4요소로 구성된다.

  $C = (P, T, I, O)$

  $P = \{1, 2, \ldots, \}, \geq 0$

  $T = \{1, 2, \ldots, \}, \geq 0$

  $I : T \rightarrow P \ \infty$

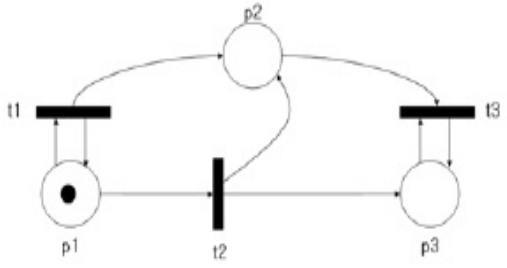
  $O : T \rightarrow P \ \infty$

페트리넷에서 나타나는 요소의 명칭과 설명은 표1과 같다. Place(P)는 원의 형태로 나타내고, Transition(T)는

직선이나 사각형의 형태로 나타낸다. I와 O에 해당하는 Edge는 화살표를 가진 선분의 형태로 표현된다. 마지막으로 Place 내부에는 토큰(Token)이 점으로 표현되어 있어서 토큰의 갯수와 토큰을 가진 Place들로 시스템의 현재상태가 표현된다. 그림4는 페트리넷으로 표현된 간단한 시스템이다[7].

[Table 1] Factor name and explanation in Petri Net

명칭	설명
Place(p)	상태나 이벤트 조건을 표현
Transition(t)	사건의 발생을 표현
Edge(e)	P와 T의 관계를 표현(입력/출력)
Token	현재 시스템 상태를 표현



[Fig. 4] Example of Petri Net

3.2 가설 설정

선행 이론과 연구[3-5, 8-10]를 바탕으로 실증연구를 위한 연구모형으로 다음과 같이 가설을 도출하였다.

H1. 서로 다른 디자인 패턴에 따라 병렬 컴퓨팅의 이해하는 바는 틀리다.

H2. 숙련도의 깊이가 깊을수록 서로 다른 디자인 패턴을 이해하기 쉽다.

3.3 연구 설계

본 연구는 서로 다른 디자인 패턴을 연구한 이를 대상으로 페트리넷을 이용 병렬 컴퓨팅기법의 이해도를 측정하는 연구이다. 연구의 실험지는 연구모형과 선행연구를 토대로 구성하였다. 표2에 숫자는 실험군 숫자이며, 프로그래밍 언어와 모델링 기법은 실험 대상자들이 알고 있는 것을 조사한 사항입니다.

[Table 2] Study Object

프로그래밍 언어	Hopes class	CUDA class	계
C	27	17	44
C++	25	12	37
C#	3	2	5
Java	18	7	25
모델링 기법			
Data Flow Diagram (DFD)	28	6	34
Object Modeling Language (OML)	7	3	10
Unified Modeling Language (UML)	15	6	21
Petri Nets (PNs)	6	1	7

2010년 11월 19일과 12월 13일 양일간 프로그래머를 대상으로 실험을 수행하였다. 이를 위해 학부, 석사, 박사생 56명을 대상으로 페트리넷의 기본 이해와 실습을 3시간에 걸쳐 선행 학습시켰고 1시간의 실험실습을 하였다. 숙련도는 학부, 석사, 박사생을 초, 중, 고급으로 분류하였다. 이들이 치른 실험지는 별첨(외국인 포함으로 영문작성)과 같다. 실험1의 두 가지 문제는 서술된 말을 바탕으로 해석하여 실험자들이 페트리넷을 이용, 실제 모델을 그릴 수 있는 지에 대한 질문이다. 예를 들어, 실험1의 1번 문제에는 “The consuming process must be in sequence with respect to the production process.”와 같이 제약조건을 삽입, ordering/ data/ task의 선험적 이해도를 페트리넷을 활용하여 표현하게 하였다. 이를 통해 태스크 중심의 언어를 알고있는 CUDA의 실험군과 Data flow 중심의 HOPES 실험군의 이해도가 어떻게 틀린지를 확인할 수 있다. 실험2의 두 가지 문제는 실험1의 질문보다는 난이도가 낮은 페트리넷 모델을 보여주고 각 모델의 역할이 무엇인지를 질문하여 위와 같은 ask 중심의 언어를 알고 있는 CUDA의 실험군과 Data flow 중심의 HOPES 실험군의 이해도가 어떻게 틀린지 확인 할 수 있다.

가설 검증을 위해 설문지를 배포하였고 주요 항목들은 Likert 7점 척도를 사용하여 측정되었다.

4. 실증분석

4.1 가설검증

독립표본 t검증이란 두 집단의 차이를 분석하는 것을 말한다. t검증의 기본원리는 두 집단의 분산과 두 집단의 분산을 합한 값을 이용해서 두 집단의 평균이 통계적으로 유의미한 차이가 있는가를 검증하는 것이다. 한편 표

본집단의 각 관측값이 평균으로부터 떨어져있는 정도인 ‘분산’이 두 집단에 있어 동일함을 ‘분산의 동질성’이라고 하는데 이 기본 가정이 t검증 과정에 앞서 검증된다[11]. 이 연구에서는 HOPES와 CUDA사용자를 대상으로 하였고 그 숙련정도를 학력별 연구원을 대상으로 주어진 문장을 통해 페트리넷을 묘사하는 능력과 페트리넷이 주어준 후 그 모형을 이해하는 척도를 검정하였다.

실험데이터의 분석은 두 가지 경우로 진행되었는데 학습량에 따라서 서로 다른 병렬 디자인패턴을 이해하는 정도가 다를 것이란 가설1과 각자의 영역에 대한 디자인 패턴은 생소한 페트리넷을 이용하더라도 쉽게 이해할 수 있을 것이란 가설2를 검정하였다.

실험자가 습득하고 있는 서로 다른 디자인 패턴에 따라 병렬프로그래밍의 이해는 틀릴 것이라는 가정을 검증하기 위해 표본의 평균치에 대한 차이를 검증하는 통계기법인 독립표본 t 검증을 실시하였다. 디자인 패턴에 따른 독립표본 t검증 결과는 유의확률(p-값)이 네 가지 실험 전부 0.05를 넘어(p-값 >α) 등분산이 가정됨을 채택하였다. 그에 따른 평균의 동일성에 대한 t검정은 네 가지 질문 모두 0.05를 넘는 값으로 나왔다. 이에, **H1(서로 다른 디자인 패턴에 따라 병렬컴퓨팅의 이해하는 바는 틀리다)** 가설은 기각 되었다.

숙련도에 따른 검증에서 실험자가 습득하고 있는 서로 다른 디자인 패턴의 숙련도에 따라 병렬프로그래밍의 이해는 틀릴 것이라는 가정을 검증하기 위해 또한 독립표본 t 검증을 실시하였다. 독립표본 t검증 결과는 한 가지(S2)를 제외한 세 가지 실험의 유의확률(p-값)이 0.05를 넘어(p-값 >α) 등분산이 가정됨을 채택하였다. 이에, **H2 (숙련도의 깊이가 깊을수록 서로 다른 디자인 페턴을 이해하기 쉽다)** 가설은 기각 되었다.

가설 1과 가설 2에 대한 독립표본 t-검증 통계의 결과는 Table 3과 같다.

4.2 검증 요약 및 분석

첫째, 이번 연구의 전체적인 면을 살펴보면, 먼저 서로 다른 디자인 패턴을 숙련한 사람들 속에서 병렬 프로그래밍의 이해도는 틀릴 것이라는 가설은 채택하기 힘든 값을 보였다. 이를 살펴보면 p-value값을 사용한다 하더라도 표본집단을 추출하는데 현재까지는 병렬프로그래머라고 할 수 있는 대상을 실험 군으로 넣기에는 쉽지 않은 일이었기 때문이라고 본다. 또한 이들에게 익숙하지 않은 디자인패턴인 페트리넷을 짧은 시간 동안에 숙지시키고 실험을 한 부분에서 보완이 필요할 것이라고 본다. 비교대상을 확대하고 좀 더 깊이 있는 교육이 동반된다면 다

[Table 3] The statistical results of hypothesis testing

디자인 패턴에 따른 검증(H1)		Levene의 등분산 검정		평균의 동일성에 대한 t-검정		
		F	유의확률	t	자유도	유의확률 (양쪽)
S1	등분산이 가정됨	0.390	0.535	-1.388	58	0.170
	등분산이 가정되지 않음			-1.346	35.132	0.187
S2	등분산이 가정됨	0.010	0.919	-1.760	58	0.084
	등분산이 가정되지 않음			-1.714	35.527	0.095
S3	등분산이 가정됨	0.036	0.851	-1.166	58	0.243
	등분산이 가정되지 않음			-1.145	36.376	0.260
S4	등분산이 가정됨	0.857	0.358	1.155	58	0.253
	등분산이 가정되지 않음			1.100	33.595	0.279
숙련도에 따른 검증(H2)		Levene의 등분산 검정		평균의 동일성에 대한 t-검정		
		F	유의확률	t	자유도	유의확률 (양쪽)
S1	등분산이 가정됨	0.355	0.553	-1.934	58	0.058
	등분산이 가정되지 않음			-2.017	46.243	0.049
S2	등분산이 가정됨	8.251	0.005	-1.391	58	0.169
	등분산이 가정되지 않음			-1.258	31.182	0.218
S3	등분산이 가정됨	3.850	0.055	-3.052	58	0.003
	등분산이 가정되지 않음			-3.323	51.614	0.002
S4	등분산이 가정됨	0.031	0.861	-2.052	58	0.045
	등분산이 가정되지 않음			-2.016	39.089	0.051

른 결과 값을 보일 수 있다는 것을 제시하였다.

둘째, 숙련도의 깊이가 깊을수록 서로 다른 디자인 패턴을 이해하기 쉽다는 가설도 기각되었다. 그러나 이 중 S2의 값은 99%의 신뢰도를 보였다. 이는 다시 말해, 어떠한 병렬프로그래밍을 배웠던 사람이라도 공통분모의 이해도는 일치하고 좀 더 특화된 영역에서 돌출 될 수 있는 여지가 있다는 점을 도출할 수 있었다.

5. 결론

5.1 요약 및 시사점

이 연구에서는 MPSoC (Multiprocessor System-on-Chip) 환경에서의 병렬프로그래밍 기법의 분류와 그 분류에 따른 서로 다른 학습자의 인지도를 페트리넷을 활용하여 검증 및 실증분석 하였다. 그중 CUDA와 HOPES를 습득한 실험자를 대상으로 서로 다른 디자인 패턴을 숙련한 대상은 큰 테두리의 병렬프로그래밍 기법에서 서로의 이해도에 차이를 둔다는 가설과 숙련도의 차이에 따라 페트리넷을 적용했을 때 차이를 보일 수 있다는 두 가지 가설의 검증이었다. 그 결과 통계학적으로 신뢰할만한 수준에 못 미치는 결과(가설1, 가설2)를 얻었다. 하지만 전체적인 면에서 보았을 때 이번 연구의 두 가지 큰 축 중의 하나인 “숙련도의 깊이가 깊을수록 서로 다른 디자인 패턴을 이해하기 쉽다(H2)”에서 의미 있는 값으로 연결될 여지를 발견할 수 있었다(S2). 이는 병렬프로그래밍의 큰 테두리에서 이를 각각 DATA중심, TASK중심, Hybrid 형으로 나누고 각 영역을 대표하는 디자인패턴의 습득자를 비교해 보았다는 점에서 또 다른 의미를 찾을 수 있었다.

5.2 연구 한계 및 과제

먼저, 모집표본을 추출하는데 어려움이 있었다는 점이다. 아직 현업 및 대학에 완전한 커리큘럼이 없다보니 연구단계로만 진행되는 병렬프로그래밍 디자인패턴에 관한 숙련자가 부족한 편이다. 또한 어느 정도의 깊이 있는 공부가 필요한 시점에서의 신뢰성 있는 실험이 필요하지만 그 기대치에는 실험자로부터 신뢰성 있는 데이터를 확보하는데 어려움이 있었다. 또한 실험에 필요한 기초지식인 페트리넷에 관한 실험자의 숙련도가 기대치보다 낮아 실험을 진행하는데 그들이 어려움을 표현하였다.

이러한 한계점을 극복하기 위하여, 실험 대상자를 전문가 급으로 한정한다면 제시한 가설에 대한 정교한 검증을 할 수 있을 것으로 본다. 또한 페트리넷에 대한 이론적 지식습득도 좀 더 깊이 있게 진행한다면 정교한 검증을 할 여지가 있다. 또한 CUDA와 HOPES 이외의 현재 많이 사용되고 있는 OpenCL과 OpenMP의 숙련자를 향후 추가한다면 좀 더 의미 있는 연구가 되리라 본다.

References

[1] Blake G. et al. "Evolution of thread-level parallelism in desktop applications". In Proceedings of the 37th annual international symposium on Computer architecture (ISCA'10), 2010.

[2] Chandy K. M. "Concurrent program archetypes". In Proceedings of the Scalable Parallel Library Conference, 1994.

[3] Massingill B. L. "A structured approach to parallel programming", Technical Report CS-TR-98-04, California Institute of Technology, 1998 (Ph.D. thesis.).

[4] Massingill B. L. and Chandy K. M. "Parallel program archetypes", Technical Report CS-TR-96-28, California Institute of Technology, 1996.

[5] Massingill B. L. et al. "Reengineering for Parallelism: An Entry Point for PLPP for Legacy Applications", Proceedings of the Twelfth Pattern Languages of Programs Workshop (PLoP 2005), 2005.

[6] "MPI: A message-passing interface standard". International Journal of Supercomputer Applications and High Performance Computing, Vol. 8 pp. 3-4, 1994.

[7] Murata T. "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, Vol. 77, No. 4, 1989.
DOI: http://dx.doi.org/10.1109/5.24143

[8] Cole M. I. Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, 1989.

[9] Coplien J. O. and Schmidt D. C., editors. Pattern Languages of Program Design, pp. 1-5. Addison-Wesley, 1995.

[10] Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[11] Shin M. C. Basic Statistics for Business and Economics. Changminbooks, 2010. pp. 319-350.

Appendix.

Experiment 1

Problem 1.

Create a Petri Net (PN) that models two concurrent processes, the producer and the consumer, who share a common, unbounded buffer. The producer produces objects that are put into a buffer from which they can be removed and consumed by a consumer. The consuming process must be in sequence with respect to the production process. If the consumer is ready to consume and an object is in the buffer, a transition can fire and remove the object from the buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e. removing it from the buffer) one piece at a time.

Problem 2.

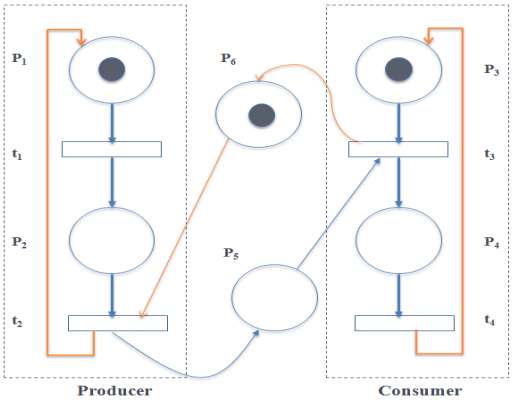
Create a PN that models the operation of a piece of equipment that is either in an on or off state: the first place means that it is operational, and the second place means that it is non-operational. Transitions fire when there is either a failure or repair. Therefore, if the equipment fails, it can later be repaired, after which it is operational again. The system has K identical components in parallel redundancy, so that they can serve as a backup in case of a component failure.

Experiment 2

Problem 1.

Consider the following Petri Net.

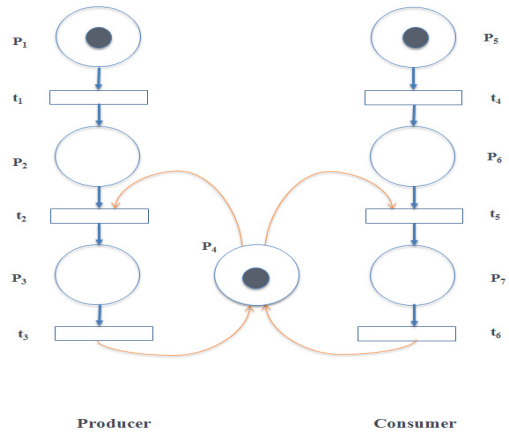
1. What does place P6 model?
2. What does place P5 model?
3. What do the number of tokens in P5 and P6 represent?
4. What situation does the initial token that is assigned to P6 model?



Problem 2.

Consider the following Petri Net.

1. Places P2 and P6represent resources requesting access to ____ and ____ respectively.
2. What is the purpose of place P4?
3. If both P2 and P6 are both marked, what happens with transitions t2 and t5?
4. What happens when t2 or t6 are fired?
5. What is the name given for this type of problem?



한 지 연(Jiyeon-Han) **[정회원]**

- 2009년 2월 : 중앙대학교 정보시스템학과 (학사)
- 2011년 8월 : 한양대학교 정보기술경영학과 (공학석사)
- 2011년 1월 ~ 현재 : SK C&C 매니저

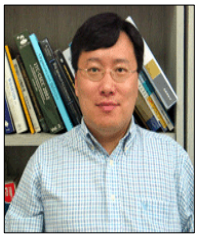
<관심분야>
컴퓨터 네트워크, IT 서비스, 시스템 감사

안 종 창(Jongchang-Ahn) **[정회원]**

- 1994년 2월 : 고려대학교 경제학과 (학사)
- 2002년 8월 : 세종대학교 소프트웨어대학원 인터넷S/W학과 (공학석사)
- 2007년 8월 : 한양대학교 일반대학원 정보기술경영학과 (공학박사)
- 1996년 1월 ~ 2010년 8월 : (주) 데이콤, SK브로드밴드 매니저
- 2010년 9월 ~ 현재 : 한양대학교 정보시스템학과 조교수

<관심분야>
지식경영, SNS 행태분석, 시스템감사

이 욱(Ook-Lee) **[정회원]**



- 1987년 2월 : 서울대학교 계산통계학과 (학사)
- 1989년 6월 : Northwestern대학교 전산학과 (전산학석사)
- 1997년 1월 : Claremont대학교 경영정보학과 (경영정보학박사)
- 1997년 9월 ~ 2002년 2월 : 한성대학교, Queensland대학교 등 교수
- 2002년 3월 ~ 현재 : 한양대학교 정보시스템학과 교수

<관심분야>
정보시스템, IT 분야 철학/행태/응용