技術動向

iAPX432를 이용한 Multiprocessing

全 昌 浩
(成均館大 工大 助教授)

■ 차 례 ■

1 서 론

1971년의 8-bit 8008부터 시작하여 microprocessor의 개발을 주도해 온 Intel社는 1981년에 iAPX 432[1]의 개발을 공표함으로써 32-bit VLSI microprocessor시대의 막을 또 먼저 열었다.

iAPX 432가 가지는 의의는 그 word length라든가 집적도를 보는 기술적인 측면에서 보다도 종래의 microprocessor들의 판에 박힌 hardware의 개념을 초월하여 과감한 architecture의 혁신을 시도했다는 데 있다. 다시 말하자면, 통상 operating system level의, 결코 간단하지 않은 software를 통해서만 실현 가능했던 기능들 중 특히 다음의 것들을 적어도 부분적으로 microprogram化하여 chip의 hardware에 포함시켰다는 것이다.

- Multiprocessing mechanism : process management, interprocess communication, interprocessor communication
- Memory management : heap organization
- Capability-based addressing
- Fault handling.

이것은 장래 microprocessor를 이용한 multiprocessor system구현을 손쉽게 하고 힘든 software개발의 부담을 극소화하고자 한 기본 설계 방침이 실현된 결과였다.

그러나 그 당시 기술로는 이 모든 기능을 하나의 chip속에 담을 수가 없었고 결국 I/O processing기능까지 포함하여 세 개의 chip으로 분리하여 만들게 되었다. 즉, iAPX 432는 instruction을 fetch하고 decode하는 기능을 담고 있는 43201과 address decode와 연산기능을 담고 있는 43202, 그리고 I/O channel 역할을 하는 43203, 이렇게 세개의 chip으로 구성되어 있다.

본 稿에서는 multiprocessing을 용이하게 해주는 기능을 갖춘 432 processor들이 실제 어떻게 system으로 구성될 수 있는 지를 예를 통하여 알아 보고자 한다. 각 432 processor들의 개별적 구성이나 자세한 동작원리에 관심이 있

는 독자는 해당되는 chip의 reference manual 이나 참고문헌 (2)를 참조하기 바란다.

본 稿의 제2장에서는 432 processor를 이용한 기본 Multiprocessing system의 예로 Intel 社의 System 432/600의 구성을 소개한다. 제3장에서는 System 432/600을 program해 본 경험을 토대로 multiprocessor system의 programing에 관하여 몇가지 문제를 간략하게 다룬다.

2 System 432/600의 개요

서론에서 간략하게 언급된 바와 같이 432 processor들은 애초부터 multiprocessing에의 응용을 염두에 두고 설계되었다. 이 章에서는 432 processor를 사용한 multiprocessor system의 실예인 System 432/600 (이하 줄여서 System 432라 칭한다.)의 구성을 살펴 보기로 한다.

2.1 Hardware 구성

System 432는 Intel社가 제작한 shared memory multiprocessor system이다. 그림1은 board level로 그려진 hardware구성도이다. 보이는 바와 같이 System 432는 두 개의 subsystem으로 이루어져 있다. 즉, 그림에서 점선 상단에 보여진 central subystem과 하단에 보여진 peripheral subsystem이다.

Central subsystem은 user program을 수행하는 부분으로 다음의 네 가지 성분으로 구성되어 있다.

a) General Data Processor(GDP) : 일반적인 data processing기능을 수행하는 것으로서 여기에 iAPX 43201과 43202 두 chip이 포함되어 있다. 하나의 System 432에 도합 다섯개까지의 GDP가 사용될 수 있다.

b) Central Memory : central subsystem 내의 GDP와 peripheral subsystem의 Interface Processor(IP)가 공동으로 사용할 수 있는 shared memory로써 1/2 Mega byte까지 확장될 수 있는 RAM이다.

c) Memory Controller(MC) : 여러 개의 processor들이 memory를 access하는 것을 제어해 준다.

d) Interface Processer Link(IPL) : I/O 기능을 수행하는 peripheral subsystem과의 interface를 담당하고 있다. 그림에서 꼬불어진 선으로 나타내진 것처럼 peripheral subsystem의 IP와 cable로 직접 연결되어 이 IPL-IP쌍은 System 432에 있어서 I/O Processor를 형성하게 된다.

이상의 네 가지 기능적 성분은 time shared bus인 central subsystem backplane에 연결되어있는데 processor들 사이에 bus사용은 round robin원칙에 입각하여 교대로 허용되고 있다. 여기서 processor라 함은 GDP와 IPL-IP쌍에 의해 형성되는 I/O processor를 총칭한다. 그래서 그림 1에 보여진 대로의 구성에서는 도합 여섯 개까지의 processor가 존재할 수 있는 셈이다. 그러나 이 여섯이라는 한정된 숫자는 central subsystem backplane에 마련된 board slot 수가 한정되어 있기 때문이라는 것과 backplane을 확장시켜 board slot의 수를 더 늘린다면 이론적으로 더 많은 수의 processor들을 한 system내에 사용할 수 있다는 것을 참고로 알아 주기 바란다. (processor의 수가 많아질 수록 증가하는 contention으로 인하여 shared-bus, shared-memory system에서는 어느 정도의 수보다 더 많은 processor를 사용할 경우 전체 system의 성능이 오히려 감소한다는 사실은 감안해야 할 것이다.)

Peripheral subsystem은 System 432에서 I/O processing을 담당하는 부분이다. 그러나, 그림에서 우리는 그 자체가 하나의 통상적인 microcomputer system 임을 볼 수 있다. 즉, 그 자체의 CPU, memory, I/O device controller 등이 central subsystem bus와는 별개인 peripheral subsystem bus에 연결되어 있다. (지금 소개되고 있는 System 432 에서는 8086이 CPU로 사용되고 있고 bus는 MULITBUS이다.) 이에 더하여 System 432의 전체 구성 상 I/O

기능을 담당하는 보조 system으로서 필요로 하는 Interface Processor(IP)를 포함하고 있다. iAPX 432 processor중 세 번째인 43203은 바로 이 IP에 주축이 되는 chip이다. 위에서 말한 대로 IPL과 더불어 이 IP는 central memory의 내용을 System 432의 외부로 내보내거나 외부로부터 data를 central memory로 집어 넣을 수 있는 입·출력 통로를 제공하고 있는 것이다.

System 432를 위한 I/O기능을 수행하는 데 있어서 IP는 peripheral subsystem 내에서는 64Kbyte에 해당하는 memory space — 이를 IP window라고 부르며, 그 starting address는 jumper wire로 결정할 수 있다. — 를 점유하면서 central memory의 data segment와 mapping 시켜 주고 있다. 다시 말하면, System 432 측에서 본 I/O processing은 peripheral subsystem의 CPU가 peripheral memory번지 중에서 IP window에 속하는 곳에다 data를 써 넣거나, 그런 번지로 부터 data를 읽어 냄으로써 실행되며, 이 IP window와 central memory내의 어떤 segment와의 matching 관계를 IP가 유지시키는 것이다. 이에 대한 더 상세한 설명은 IP에 관한 reference manual을 참조하기 바란다.

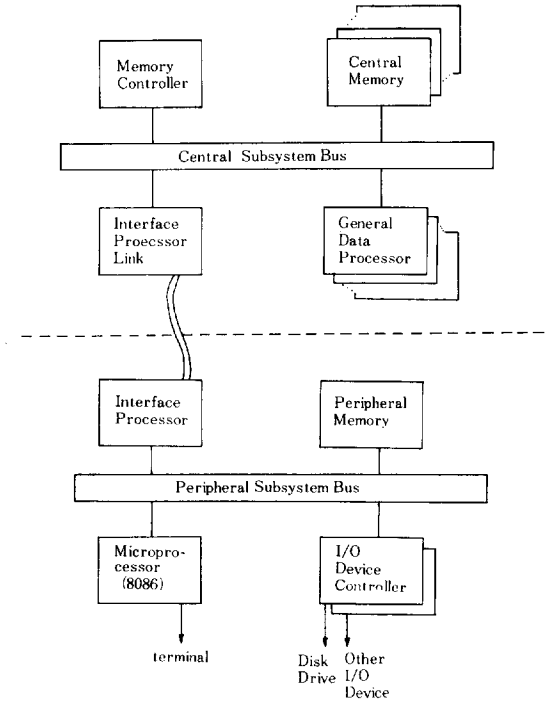

그림 1. System 432/600의 구성도

Peripheral subsystem을 통하여 운용될 수 있는 I/O device로는 terminal을 비롯하여 disk 등 어떤 형태의 주변장치라도 그에 맞는 controller와 driver software만 갖추어져 있으면 가능하다. System 432의 operator console도 I/O device의 일종으로 동작하므로 이런 구성의 peripheral subsystem을 통해 붙여 진다. 만약에 I/O processing 능력이 더 많이 요구된다면 그림 1에 보인 것과 같이 구성된 peripheral subsystem을 더 늘릴 수 있다. 단, 이때 요구되는 점은 하나의 peripheral subsystem당 한 개의 IPL-IP 쌍이 필요하다는 것이다.

2.2 Program의 개발과 수행

이상에서 System 432는 iAPX 432가 주축이 되어 user program을 수행하는 central subsystem과 보통의 16-bit microcomputer system인 peripheral subsystem으로 구성됨을 알았다. 이제 이야기 하고자 하는 것은 이 System 432에 수행될 응용 program이 실제 어떻게 개발되고 수행되는 가 하는 것이다.

System 432를 program하는 언어는 Ada* 이다. 즉, iAPX 432는 microprocessor로 불리우지만 그 user는 종래의 microprocessor 들처럼 assembly language로 program을 작성하는 것이 아니라 high level language, 특히 Ada로 작성한다. Ada가 iAPX 432의 program·mining language로 된 이유는 iAPX 432를 설계할 때부터 그 기본 목표로 삼은 object-oriented architecture의 실현, multiprocessing mechanism의 hardware化 등의 착상이 마침 비슷한 시기에 진행된 Ada language의 개발에 근본적으로 강조되었던 access protection, tasking 등의 요구 사항이 거의 일치하거나 근사했기 때문이다.[3] 그러나, System 432는 Ada program을 수행할 수는 있으되 그 자체에 program 개발

*) Ada는 美 국방성의 등록상표임.

에 필요한 도구가 갖추어 져 있지 않다. 그림 2는 System 432에 수행시킬 user program을 개발하고 수행시키는 데 필요한 설비를 보여주고 있다. User program은 다른 host computer에서 executable code로 준비된 다음 peripheral subsystem을 경유하여 central memory로 옮겨져 수행된다. 그림 2를 참조하면서 그 과정을 순서대로 말하자면 다음과 같다.

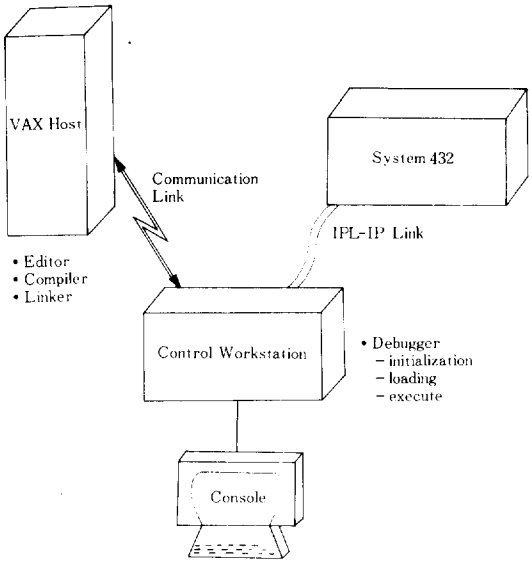

그림 2. System 432 Programming Environment

먼저 user는 host computer의 editor를 이용하여 Ada source program을 작성한 다음, 역시 host computer에 있는 Ada cross compiler와 linker를 이용하여 object code를 만든다. (이때 object code에는 compile된 use program과 미리 compile되어 있는 System 432 전용 operating system module이 합쳐 지게 된다.)

그리고 난 후 control workstation의 command를 사용하여 그 object code를 host computer로부터 control workstation의 memory로 download한다. 그리고 역시 control workstation에 있는 debugger를 사용하여 download 된 object code를 System 432의 central memory에 load 시키고 execute하라고 하는 command까지 준다.

여기서 control workstation은 System 432의 console 역활을 하는 제 2의 peripleral subsystem의 위치를 차지하고 있는 셈이다. 또한, control workstation의 terminal은 user program이 수행되고 있는 동안 standard I/O device로써 system message나 program 수행상 필요한 data의 입·출력이 가능한 장치도 된다.

3 System 432에서의 Multitasking

앞 章에서 본 바와 같이 System 432 그 자체만으로는 독립된 multiprocessng system이 되지 못한다는 것이 아쉬운 점이다. 그렇긴 하나 이제 막 보급이 가속되고 있는 Ada language의 가장 두드러진 특징 중의 하나인 multitasking을 진정한 의미에서 실행해 내는 유일한 System이라는 점에서, 또 그것이 어느 연구실 내의 실험용이 아니라 시장에서 누구라도 살수 있는 system으로 실현될 수 있는 가능성을 실증해 보였다는 점에서 System 432의 진가는 충분히 음미해 볼 만하다. 본 章에서는 이 System 432에 Ada multitasking program을 수행시켜 본 결과를 보임으로써 오늘날 VLSI microprocessor를 사용한 multiprocessor system을 program하는 데 도움이 될 몇 가지 실험적 자료를 제시하고자 한다.*

3.1 Procedure와 Task간의 선택

Ada language에 정의된 procedure와 task는 일련의 Program Statement들을 하나의 단위, 즉, 한 곳의 시작점(*begin*)과 한 곳의 종료점(*end*)이 존재하는 program module을 만드는 데 이용될 수 있다는 공통점을 갖고 있다. 그러나, semantic 차이는 현저하다. Procedure는 순차적으로 수행되는(sequential) 단위인 반면 task는 동시에 수행되는(concurrent) 병열 단위이다. 그러면 program하고자 하는 algori-

*) 이 章에서 다루는 내용의 일부는 참고문헌 4)에 게재 된 바 있음.

thm에 꼭 같은 code로 반복, 또는 되풀이 해야 할 부분이 있을 때 System 432와 같은 류의 multiprocessor system에서는 어느 편을 이용하는 것이 더 효율적일 것인가? 일차적으로 생각하기를 반복될 operation사이에 순서 의존도가 없다면 당연히 task를 이용하는 편이 유리할 것이라고 할 지 모른다. Program의 효율성을 주어진 계산량을 다 처리하는 데 소요되는 총 시간의 관점에서 논한다고 가정할 때 반드시 그렇지 만은 않음을 보여 주는 결과가 그림 3과 4에 주어졌다. 두 그림에 보인 것은 일정량의 계산을 수행하는 데 procedure만으로 구성된 program을 실행시켰을 때 걸린 시간과 task만으로 구성된 program을 실행시켰을 때의 시간을 graph로 나타낸 것이다. X축은 System 432내의 processor(GDP)수, Y축은 실행에 소요된 총 시간을 나타낸다. 그림 3은 각 module (procedure 이거나 task 이거나)의 수행시간이 100 msec로 동일하게 coding된 경우이며, 그림 4는 그것이 6 sec인 경우이다. 두 그림에서 실선으로 나타 낸 graph는 task로 작성된 program의 결과이고 점선으로 나타 낸 graph는 procedure로 작성된 program의 결과이다. 또, △표로 이어진 선은 한 program속에서 30번의 procedure call을 반복하는 (또는 30개의 task를 사용한)경우이며, ○표로 이어진 선 들은 20번, □표는 10번을 반복하는 경우를 각각 나타 낸다.

이 두 그림에서 볼 수 있는 것은 첫 째 각 module의 수행시간이 짧을 경우 processor 의 수가 많건 적건 간에, 또 반복 수행되는 횟 수에 관계없이 공히 procedure를 사용하는 편이 유리하며, 수행시간이 길 경우 그 반대 현상이 나타났다는 것이다. 이 현상은 task 하나를 생성시키는 데 소요되는 operating system overhead(task creation time)가 상당히 크다는 걸 말 해준다. 즉, module當 수행시간이 짧을 경우(그림 3)는 여러개의 processor가 있음으로 해서 병열 수행도가 커 지고 그에 따른 전체소요시간의 이득이 있을 수도 있으나 상대적으로 훨씬 큰 task creation overhead가 그 이득을 상쇄할 뿐아니라 오히려 전체 소요시간의 대부분을 차지 해 버림으로써 task를 사용한 경우가 더 불리하게 나타났다고 해석 할 수 밖에없다. Ada의 task는 일단 존재한다면 processor가 허용되는 한도 만큼의 병열 수행이 되나, 一群의 task들이 생성될 필요가 있을 때 그 생성과정 자체는 병열로 진행되는 것이 아니기 때

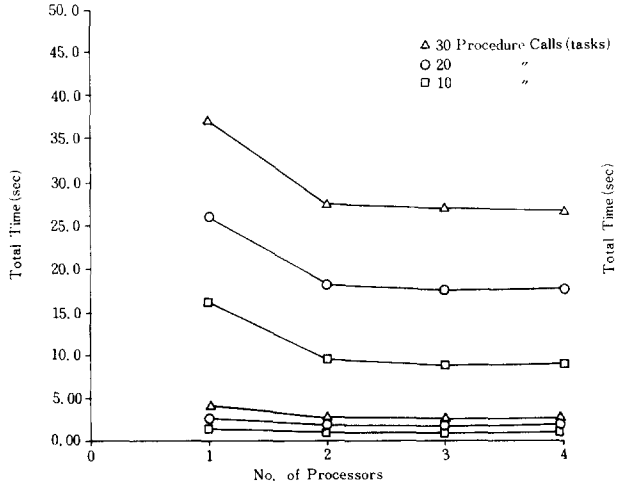

그림 3. Procedure로 작성된 program과 task로 작성된 program의 수행시간 비교 (module 당 수행시간 100 msec인 경우)

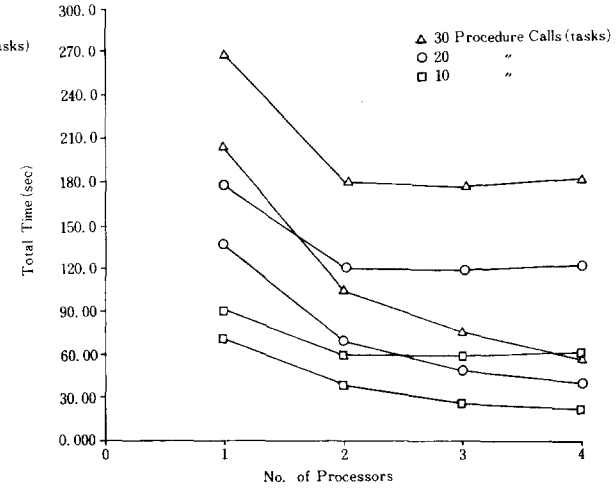

그림 4. Procedure로 작성된 program과 task로 작성된 program의 수행시간 비교 (module 당 수행시간 6 sec인 경우)

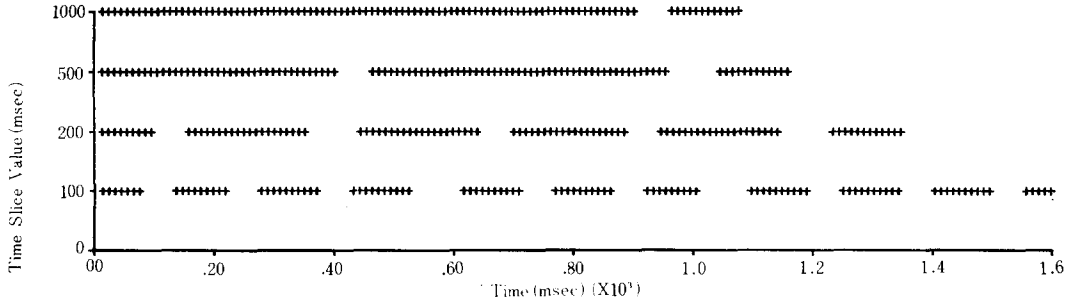

그림 5. Single task program을 single processor에 수행시킨 경우 time slice 값에 따라 변하는 execution profile

문이다. Module當 수행시간이 task 생성에 소요되는 overhead보다 상대적으로 길 때는(그림 4) 물론 task를 사용하는 편이 유리하다.

다른 한 가지 우리가 확인 할 수 있는 사실은 processor의 수가 증가할 수록 task로 구성된 program은 속도이득을 증가시키고 있는데 반해 procedure로 구성된 program은 processor가 더 존재 해도 그 효과가 보이지 않는다는 것이다. 이는 task들이 병열수행 되는데 반해 procedure 경우는 순차적으로 수행될 뿐이므로 잉여 processor가 아무 일을 하지 않고 idle하고 있을망정 전체 소요시간의 단축에 도움을 주지 못하기 때문이다.

3.2 Time Slice 값의 영향

어떤 system 내에서 processor의 수보다 많은 수의 task가 수행될 때 processor들은 어느 일정시간 동안 하나의 task를 수행한 다음 다른 task로 바꿔서 수행해 주어야 한다. (이를 task swapping이라 한다.) 이때 하나의 task가 어느 processor에 의해 수행되기 시작한 후부터 다음 task를 위해 processor를 양보해 주기까지의 시간이 time slice이며 이것은 operating system의 중요한 parameter이다. 이제 이 time slice의 값의 크기가 전체 system의 성능에 미치는 영향을 보이기 위하여 일정한 계산량을 task하나로 program한 다음 하나의 processor만 사용하여 수행시키면서 time slice 값에 변화를 주어 보았다. 그림 6는 그 수행 추적도인데 X축은 수행완료시간이며 Y축은 변화되는 time slice값을 msec로 나타낸다. 그림에서 +로 연결되어 나타나는 부분은 실제 user

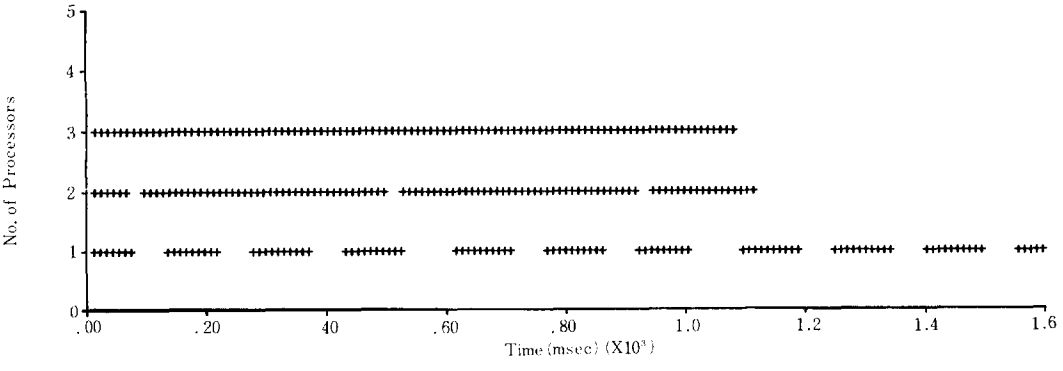

그림 6. Processor의 수에 따라 변하는 single task program의 execution profile (time slice 값은 100 msec로 고정)

program이 수행되는 시간들이고 공백으로 이어진 부분은 operating system service를 위해 program의 수행이 중지된 시간이다. User program에 주어지는 time slice값이 길면 길 수록 일정한 량의 계산을 하는 동안 하나 뿐인 processor를 operating system service에 양보하는 시간이 줄어 들고 그 만큼 전체 계산시간의 단축을 누리게 된다.

사실은 이 time slice 값은 일반적으로 user level에서는 조정할 수 없는 것이며 operating system process는 보통 user process보다 priority가 높게 주어지기 때문에 이 실험은 일반적인 여건 하에서는 별 의미를 가지지 못한다. 그러나, System 432 같은 경우는 어떤 목적에 전용으로 사용될 수 있는 것이며 그런 환경을 서는 user가 곧 system manger일 수 있는 것이다. 그런 관점에서 이 실험은 operating system orocess(예, garbage collector) 와 user process의 priority값을 동일하게 부여하고 time slice 값을 조정해 본 것이다. User task의 time slice를 크게 할 수록 총 소요시간에 나타나는 이득은 커지는 것으로 실증되었다. 흔히 garbage collector등 operating system process는 system의 정상적인 동작을 보장하기 위해 중대한 기능을 수행하는 것이므로 user task의 time slice를 조정할 수 있다 하더라도 지나치게 크게 한다면 operating system process가 필요한 때 제 기능을 수행할 수 없음으로 해서 (starvation) 전체 system의 동작이 저해 또는 마비될 위험성이 있음을 주의 해야 할 것이다.

3.3 Processor수의 영향

Time slice 값을 증가 시킴으로써 얻을 수 있는 있는 속도 이득은 결국 operating system overhead를 감축시킬 수 있다는 데 있었다. 그러나 그 방법은 항상 환영할 수 있는 방법이 아니다. 그 operating system overhead를 훨씬 안전하게 감소시키는 방법은 추가로 processor를 공급 함으로써 user program이 수행되는 도중 swapping되지 않도록 하는 것이다. System 432는 user program을 수정하지 않고 processor의 수를 증가시키거나 감소시킬 수 있는 장점을 가지고 있다 (software-transparency). 그림 6은 동일 program을 수행하는데 있어서 processor의 수를 변화 시켜가면서 얻은 추적도이다. 여기서 X 축은 총 소요시간, Y 축은 processor의 수를 나타낸다. Processor가 하나일 때보다 두 개일 때가 operating system으로부터 받은 overhead는 훨씬 적고, processor가 세 개가 되면 전혀없다는 걸 볼 수 있다. 그 결과 전체 계산에 소요되는 시간이 단축됨은 자명한 일이다.

3.4 Task수의 영향

주어진 일정량, 예를 들어, 총 150 unit의 계산량이 있을 때 program을 5 unit짜리 task 30개로 구성하느냐, 아니면 30 unit짜리 task 5개로 program하느냐 하는 것도 programmer가 결정 해야 할 문제이다. 그림 7은 이에 대한 실험결과이다. 여기서 X축은 program내의 task 수이며, Y축은 총 수행시간이다.

우선 보이는 것은 processor의 수가 한개 에서 4 개까지 증가함에 따라 계산시간은 줄어들고 있다는 것이다. 이것은 multitask processing

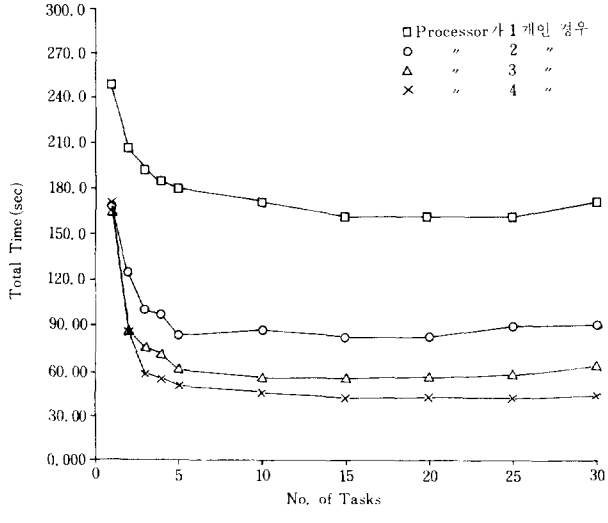

그림 7. Program 내의 task 수에 따라 변하는 전체 수행시간

에서 별로 새로운 사실이 못된다. 여기서 주목할 점은 processor의 수에 관계없이 task 수가 한개일 때의 수행시간이 가장 길고(이 때는 병열 수행의 여지가 없으며 processor의 수가 늘어 난다고 해도 operating system overhead를 감소시키는 외에 눈에 띄는 이득은 없다.) 또 program 내의 수가 처음 다섯까지 늘어나는 동안 가장 이득이 큰 반면 다섯 task이상부터는 task의 수가 증가 할 수록 오히려 미세한 역효과가 나타난다는 점이다.

이에 대한 설명은 다음과 같이 될 수 있다. Task수가 다섯 이상 일 때 우선 그 생성(creation)과 말소(deletion)에 소요되는 overhead가 전체 수행시간에서 차지하는 비중이 실제 user program의 수행에 소요된 시간이 차지하는 비중을 지배적으로 능가하기 때문이다. 둘 째 일정양의 계산을 분담하는 task의 수가 많아 질수록 하나의 task당 수행종료시간이 더 짧아질 것이고 이는 각 processor로 봐서 더 잦은 swapping을 하게 되는 결과를 낳게 되므로 그렇게 큰 비중은 아니된다 하더라도 이것 또한 overhead로 작용하는 때문인 것이다. 이 실험에서는 task 상호간의 communication은 취급되지 않았다.

4 결　　론

iAPX 432가 등장하면서 학계, 업계 모두의 비상한 관심을 모았으나 유감스럽게도 몇 가지 benchmark algorithm을 수행하는 속도를 비교한 연구[4]에서 그 성능에 대해 별로 환영할 만한 평가를 받지를 못했다. 사실 multiprocessing에의 손쉬운 응용과 capabliity-based addressing 특성을 microprogram으로 실현시킴으로써 차원 높은 architecture를 구현시키는 것을 설계의 기본 목표로 삼았던 탓으로 single processing system의 여건 하에서는 오히려 불리한 경우가 있음을 감안할 때 그 비교는 반드시 공정한 것이었다고 할 수 없기도 하다. 허나 연산기능과 instruction fetch 및 decode logic이 두개의 chip에 분리되어 있다는 결정적인 결함 때문에 기대한 만큼의 속도를 보이지 못하는 것은 부인할 수 없다.

그럼에도 불구하고 iAPX 432가 microprocessor의 발전사에 남긴 자국을 아무도 가볍게 보아 넘길 수 없는 것이다. 이런 뜻에서 그것을 이용하여 만든 multiorocessing system의 실용 예를 소개하고 그 system을 실험 연구용으로 사용해 본 경험을 토대로 multiprocessor system을 program 할 경우 참고가 될 몇 가지 실험적 data를 제시하였다.

참 고 문 헌

1) Introduction to the iAPX 432 Architecture, Intel Corp., Santa clara, CA, 1981

2) Glenford J. Myers, Advances in Computer Architecture, 2nd ed. John Wiley & Sons, Inc., 1982

3) S. Zeiger, et el., " Ada for the Intel 432 Microcomputer," IEEE Computer, Jun. 1981, pp. 47-56

4) A.P. Reeves and C.H.Jeon, "Computer Vision Task Decomposition on A Multicluster MIMD System," in Language, Architectures and Algorithms, ed. M.J.Duff, Academic Press, London(1986)

5) P.M. Hansen, et al., "A Performance Evaluation of the Intel iAPX 432," Computer Architecture News, Vol.10, No.4, Jun. 1982.