Computer Science. Tom 1. 1999 519.686:681.3.06

*Włodzimierz Funika**

NARZĘDZIA DO ANALIZY JAKOŚCI PROGRAMÓW RÓWNOLEGŁYCH OPARTYCH NA PRZESYŁANIU KOMUNIKATÓW

1. Wprowadzenie

W niniejszym artykule przedstawiono zagadnienia, związane z analizą jakości działania programów równoległych opartych na paradygmacie przesyłania komunikatów (m.in. PVM [1], MPI [2]) oraz z budową narzędzi umożliwiających tę analizę. Informacje o istniejących narzędziach są rozproszone i przeważnie przedstawione przez autorów jednostronnie, stąd celem tego artykułu było zebranie tych informacji w jednym miejscu oraz przeprowadzenie usystematyzowanej oceny narzędzi: ich zalet i ograniczeń. Powinno to ułatwić wybór właściwego narzędzia. W podsumowaniu w oparciu o przedstawioną w pracy analizę porównawczą przedstawiono propozycję dalszych działań, zmierzających do udoskonalenia funkcjonalności narzędzi.

Podstawowym celem tworzenia programów równoległych jest minimalizacja czasu wykonania. Ważną fazą powstawania programu równoległego jest uzyskanie możliwie najwyższego poziomu jego efektywności. W tej fazie tworzenia oprogramowania pomocne są narzędzia do przeprowadzenia analizy wydajności, których podstawowym zadaniem jest umożliwienie **obserwowalności jakości działania** (*performance observability*) [3]. Na pojęcie obserwowalności składa się możliwość dokładnego wychwycenia, analizy i prezentacji informacji o zachowaniu wydajności systemu. Najczęściej chodzi tu o zrozumienie przyczyn, które sprawiają, że dany program jest wolniejszy od tego, co się zakładało w fazie jego projektowania. Narzędzia powinny umożliwiać zbieranie danych o aplikacji, systemie operacyjnym, sprzęcie oraz ich syntezę w taki sposób, aby umożliwić osiągnięcie oczekiwanych wskaźników jakości działania programu.

W miarę wzrostu mocy obliczeniowej w systemach równoległych, objętość i złożoność danych o jakości działania rośnie wykładniczo. To bogactwo informacji stanowi poważny problem i dla programisty, który ma je odwzorować na swój model funkcjonowania aplikacji, i dla narzędzi, które mają za zadanie zbieranie, przechowywanie, przetwarzanie i przedstawianie informacji o jakości działania. Ogrom danych o wydajności wiąże się z wysokim

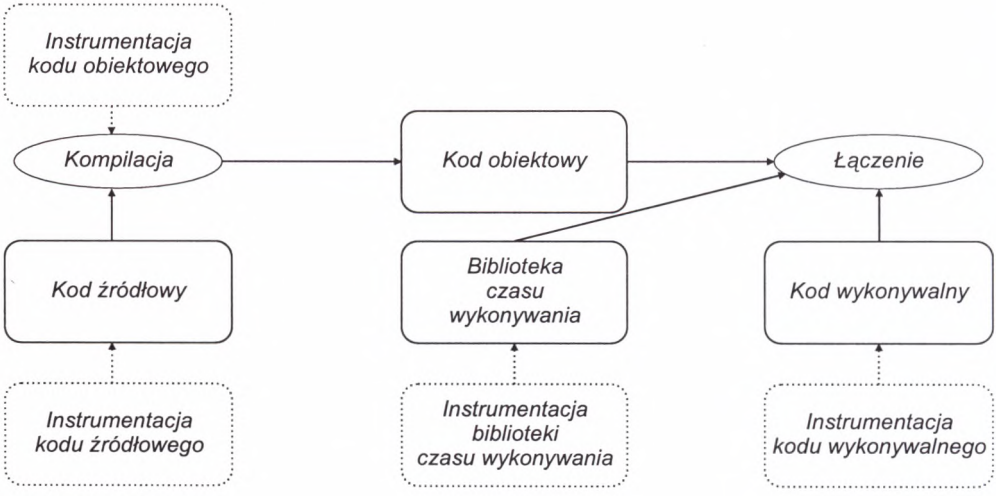
* Katedra Informatyki, Akademia Górniczo-Hutnicza, Kraków

kosztem ich zbierania. Ocenę wydajności (*performance evaluation*) można podzielić na dwie części: analizę i instrumentację. Podczas gdy instrumentacja ma za zadanie zebranie niezbędnych informacji o wykonaniu programu, analiza powinna umożliwiać ekstrakcję użytecznych informacji oraz ich interpretację.

2. Techniki analizy jakości działania programów

Złożoność i różnorodność współczesnych systemów komputerowych sprawiła, że w dziedzinie narzędzi do analizy programów równoległych istnieje obecnie cały szereg różnych technik, które można podzielić na: uzyskanie metryk jakości, mechanizmy wyszukiwania i wizualizację wydajności.

Wszystkie niezbędne dane o wykonaniu są uzyskiwane za pomocą instrumentacji, która obejmuje implementację mechanizmów zbierania danych, specyfikację i filtrację zbieranych danych. Są cztery podejścia do instrumentacji systemów komputerowych: instrumentacja kodu źródłowego aplikacji, instrumentacja kodu obiektowego, instrumentacja biblioteki czasu wykonania oraz instrumentacja kodu wykonywalnego aplikacji. Na rys. 1 są przedstawione relacje pomiędzy technikami instrumentacji a fazami przetwarzania programu.



Rys. 1. Instrumentacja programu a fazy przetwarzania programu

Instrumentacja może być wykonana statycznie, np. na kodzie źródłowym przed kompilacją programu, lub dynamicznie w trakcie wykonania programu poprzez modyfikację kodu wykonywalnego.

Mechanizmy zbierania danych mogą być zaimplementowane w postaci programowej, sprzętowej i hybrydowej. Zbieranie danych jest procesem realizowanym przez system monitorujący.

Istnieją trzy podstawowe techniki zbierania danych:

1. profilowanie (*profiling*), pozwalające na określenie czasu, spędzonego w poszczególnych fragmentach programu (*profiles*); najbardziej rozpowszechniona technika, często wystarczająca do zrozumienia zachowania programu;
2. mierzenie czasu/zliczanie (*timing/counting*) służy do pomiarów czasu (timings) lub ilości wystąpienia poszczególnych zdarzeń (counts), realizowane przy pomocy środków programowych lub sprzętowych;
3. rejestracja śladu zdarzeń (*event tracing*), która w istocie obejmuje wszystkie ważne z punktu widzenia programisty zdarzenia oraz zawiera w sobie dane, na podstawie których można uzyskać liczniki (*counts*), pomiary czasu (*timings*) oraz profile (*profiles*).

Techniką generującą największą ilość danych jest rejestracja śladów zdarzeń. W celu zredukowania objętości danych oraz przekształcenia ich w postać akceptowalną dla narzędzi wizualizacyjnych, stosowane są algorytmy przekształcenia danych.

W celu zmniejszenia zakłócenia badanej aplikacji oraz objętości zbieranych danych, niektóre podejścia stosują filtrację zdarzeń za pomocą języka specyfikacji zdarzeń, jak np. EDL [5].

Dane o jakości działania programów są informacją wielowymiarową zarówno przed, jak i po przekształceniu danych. W zależności od potrzeb analizy wymiarowość może być zmniejszona do wielkości skalarnych, najbardziej rozpowszechnione są badania 2- i 3-wymiarowe.

Rozróżniane są narzędzia interaktywne z interfejsem graficznym i narzędzia *automatyczne*, ukryte na niskich poziomach hierarchii systemu komputerowego. Są narzędzia, które pozwalają tylko na obserwację zachowania się programu oraz takie, które umożliwiają manipulowanie przebiegiem wykonania programu. Narzędzia, które mogą funkcjonować równocześnie z obserwowanym programem, nazywane są narzędziami *on–line*, narzędzia typu *off-line* przetwarzają dane po wykonaniu programu.

3. Kryteria oceny narzędzi

Ze względu na złożoność i wieloaspektowość funkcjonowania narzędzi do analizy programów równoległych, wybór najbardziej przydatnego z nich do konkretnego zastosowania powinien uwzględniać szeroką gamę kryteriów. Przedstawiony poniżej zestaw kryteriów został opracowany na podstawie studiów literaturowych [6, 7, 8] oraz badań własnych autora.

- Jakość funkcjonowania (*performance*) obejmuje:
  - Dokładność (*accuracy*) – narzędzie jest uważane za dokładne, o ile nie powoduje zbyt dużych zmian w zachowaniu i wskaźnikach czasowych monitorowanego programu.
  - Efektywność (*efficiency*) – narzędzie powinno być dosyć szybkie w wykonaniu – mała szybkość nie sprawia, że jego użytkowanie staje się nieefektywne.
  - Skalowalność (*scalability*) – oznacza, że narzut monitorowania rośnie w sposób akceptowalny przy rosnącym rozmiarze systemu i rozmiarze zadania.
- Zakres możliwości (*capabilities*) obejmuje:
  - Ogólność (*genericity*) – elementy składowe narzędzia mogą być ponownie użyte do innych operacji.
  - Personalizacja (*personalization*) – oznacza nadanie kontroli użytkownikowi nad pewnymi parametrami narzędzia i możliwość łatwego dostosowania do potrzeb użytkownika.

- Interaktywność (*interactivity*) – oznacza podtrzymywanie interakcji pomiędzy użytkownikiem a składowymi narzędzia. Chodzi tu o sposób przedstawiania danych i reakcje ze strony narzędzi przedstawiających te dane.
- Moc (*power*) – jest stopniem łatwości, z którą narzędzie może manipulować niezbędnymi danymi przy analizie wydajności, czyli śladem wykonania i pochodnymi z niego danymi.
- Przyjazność dla użytkownika (*user–friendliness*) – polega na zmniejszeniu ręcznego manipulowania ze strony użytkownika.
- Abstrakcja (*abstraction*) – narzędzie powinno umożliwiać wgląd w dane na różnych poziomach abstrakcji. Poziomy abstakcji powinny obejmować od najniższego tzn. przesyłania komunikatów do najwyższego tj. do poziomu algorytmu (programu źródłowego).

– Wszechstronność (*versatility*) obejmuje:
  - Heterogeniczność (*heterogeneity*) – oznacza, że narzędzie może funkcjonować w całości lub w postaci swoich składowych na wszystkich platformach konfiguracji w warunkach heterogeniczności wykonania.
  - Interoperabilność (*interoperability*) – oznacza, że narzędzie jest oparte na otwartych interfejsach i odpowiada istniejącym standardom.
  - Przenośność (*portability*) – oznacza, że narzędzie może funkcjonować na różnych architekturach równoległych.
  - Rozszerzalność (*extensibility*) – oznacza, że narzędzie umożliwia rozbudowę analizy o nowe możliwości, np. o manipulowanie nowymi typami danych i nowymi widokami.
– Dojrzałość (*maturity*) – obejmuje odporność na błędy, jakość dokumentacji i poziom podtrzymywania przez autorów lub producenta.
– Łatwość użytkowania (*ease of use*) – oznacza łatwość zrozumienia interfejsu użytkownika i łatwość jego używania przez typowego użytkownika.

4. Ocena narzędzi do analizy jakości programów

W tym rozdziale przedstawiono szereg reprezentatywnych narzędzi i środowisk. Zakres możliwości przedstawianych narzędzi jest dosyć szeroki: od najprostszej wizualizacji przebiegu wykonania programu do bardzo złożonej wizualizacji animacyjnej, od wizualizacyjnego sposobu przedstawiania problemów jakości do technik opartych na algorytmach automatycznego wyszukiwania źródeł strat wydajności, od „językowych" technik umożliwiających dostęp do danych do personalizowanych graficznych interfejsów użytkownika. Przy przedstawianiu poszczególnych narzędzi zostaną pokazane ich silne i słabe strony oraz możliwe ograniczenia przy praktycznym ich stosowaniu. Zwięzłe charakterystyki narzędzi są przedstawione w tabeli 1.

Narzędzia są podzielone na następujące grupy: narzędzia do wizualizacji w trybie *off-line*, narzędzia do wizualizacji w trybie *on-line*, narzędzia z konfigurowalną analizą i wizualizacją, narzędzia o dynamicznej instrumentacji i narzędzia o standaryzowanych interfejsach.

Tabela 1

Charakterystyki narzędzi do analizy jakości programów równoległych

Nazwa	Środowiskowo zależne	Typ narzędzia	Generacja śladu	Tryb działania	Rodzaj wizualizacji	Sposób instrumentacji
AIMS	+	pt	au, tpt	off	g, sm, h, p, an	g, i–a, src
Annai	+	env, db	au	on, off	g, an, sm, p	i–a,obj, exe
Carnival	+	st	au	off	g, sm, p	au, obj
JEWEL	+	pt	u	on, off	sm, g, p, an, ud	src
Medea	–	pt	au, tpt	off	g, sm, h	src
Pablo	–	pt	u	off	g, an, ud	g, i–a, src
Paradyn	–	st	au, dt	on	g, sm, p, h	i–a, exe
ParaGraph	+	pt	PICL	off	g, an, sm, p	au, src, obj
SCOPE	–	pt	au, tpt	off	g, an, sm, h, ud	au, src
SIMPLE	–	pt	au, u	off	g, sm, p, ud	au, src
Tau	+	env, db	au	on, off	g, an	au, obj
TOOLSET	–	env, db, pt	au	on, off	g, an, sm, p	au, src
Upshot	+	pt	au	off	g, an, sm, p	au, obj
VAMPIR	+	pt, db	au,tpt	off	g, sm	g, src
XPVM	+	m, pt	au	on, off	g, an, sm, p	au, obj

an	animacja	i–a	interaktywny interfejs
au	automatyczna generacja	m	system monitorujący
ctl	biblioteka komunikacyjna	off, on	*off-line*, *on-line*
db	narzędzie do debugowania	p	metryki jakości działania
di	instrumentacja dynamiczna	pt	narzędzie do analizy jakości działania
dt	narzędzie dynamiczne	sm	sumaryczne statystyki
env	zestaw narzędzi	scr	kod źródłowy
exe	kod wykonywalny	st	narzędzie inteligentne
g	graficzny	tpt	narzędzie do przetwarzania śladu
h	histogramy	ud	definiowany przez użytkownika

4.1. Narzędzia do analizy i wizualizacji w trybie off–line

4.1.1. ParaGraph

Narzędzie ParaGraph (ParaGraph) [9] zostało opracowane w Oak Ridge National Laboratory (USA) na początku lat 90.

Charakterystyka. ParaGraph jako jeden z pierwszych przyczynił się do szerokiego zastosowania wizualizacji do oceny jakości działania programów równoległych. Narzędzie używa śladów wygenerowanych przez bibliotekę programowania równoległego z przesyłaniem komunikatów PICL [10]. ParaGraph pozostaje narzędziem popularnym, a duża liczba programów monitorujących posiada moduły konwersji z ich własnych formatów na format PICL.

Możliwości. Do najważniejszych zalet ParaGraph należy szeroka gama widoków: widok wykorzystania procesora, widok komunikacji w programie (space-time, animacja grafu stanów aplikacji lub macierz komunikacyjna), widoki aktywności procesów, oraz inne widoki, jak tekstowe przedstawienie śladu, statystykę lub widok ścieżki krytycznej aplikacji. Widoki komunikacji są szczególnie dobrze opracowane. Widoki ParaGraph-u są personalizowalne w globalnym trybie, można dobrać kolory lub usytuowanie procesów na widokach (różne topologie logiczne w animacjach).

Ograniczenia. Duża liczba obiektów (ponad 32) prowadzi do nieczytelności przedstawianej informacji. Dane statystyczne generowane przez ParaGraph są mało informatywne. Wizualizacje nie są interaktywne, poza tym jest bardzo trudno odnaleźć informacje z jednego widoku na drugim. ParaGraph można uzupełnić o nowe widoki, ale to wiąże się z bardzo dużym nakładem pracy. Globalne przedstawienie funkcjonowania aplikacji nie pozwala na uzyskanie dokładnych danych o jakości działania, lecz jedynie granice przedziałów danych. Dokładne dane można uzyskać dopiero na końcu symulacji śladu przy pomocy ParaGraph-u.

Podsumowanie. ParaGraph jest wysoko ceniony w środowisku analityków jakości działania, gdyż dzięki dużej liczbie widoków oraz łatwości instalacji i posługiwania się nim pozwala na szybką ocenę zachowania aplikacji.

4.1.2. Upshot

Upshot [11] powstał na początku lat 90. w Argonne National Laboratory (USA) jako narzędzie do wizualizacji wykonania aplikacji w systemie p4.

Charakterystyka. Upshot ma tylko jeden widok w formie diagramu space-time. Format śladu jest bardzo prosty. Dostarczana jest usługa umożliwiająca konwersję istniejących śladów z innych narzędzi. Może być stosowany do programów, wykorzystujących MPI.

Możliwości. Upshot pozwala na szybką budowę widoku, personalizowalnego dla badanej aplikacji i podtrzymuje przemieszczanie się do dowolnego miejsca wykonania w programie równoległym. Typy zdarzeń w śladzie odpowiadają początkowi i końcowi pobytu w tym stanie, zaś kolor odpowiada za przedstawienie stanu w trakcie wykonania. Podczas wczytywania śladu, narzędzie zaznacza zdarzenia paskami odpowiadającymi zadaniom oraz zaznacza stany. Przebieg wykonania jest podzielony na „strony" i można przechodzić z jednej strony na drugą. Upshot umożliwia łączenie informacji ze zdarzeniami w trakcie ich odtwarzania.

Podsumowanie. Bieżąca wersja Upshot ma ograniczone możliwości. Można oczekiwać, że nowa wersja umożliwi szereg nowych wizualizacji i pozwoli na prezentację interakcji procesów.

4.1.3. Aims

AIMS został opracowany w połowie lat 90. w NAS Ames [12].

Charakterystyka. AIMS jest zestawem narzędzi, pozwalających na predykcję i optymalizację jakości działania programów równoległych, które używają biblioteki PVM i MPI. Instrumentację wykonuje się na poziomie kodu źródłowego (C lub Fortran). Kod zinstrumentowany powinien być skompilowany i zlinkowany z biblioteką zawierającą funkcje formatowania i buforowania zdarzeń, które są zależne od docelowej architektury. AIMS zawiera system syn-

chronizacji zegarów i mechanizm korekcji zakłóceń powstałych ze względu na prowadzone monitorowanie. Narzędzie do korekty uwzględnia narzut, związany z wywołaniem funkcji służących do śledzenia oraz czas opróżnienia buforów do śledzenia [13].

Możliwości. Narzędzie umożliwia analizę syntaktyczną i konstruuje graf wywołań procedur i pętli. Można wskazać obiekt do zainstrumentowania (procedury, pętle, wywołania funkcji komunikacyjnych, I/O w systemie plików). AIMS łączy zalety instrumentacji kodu źródłowego z zaletami instrumentacji przy kompilacji.

AIMS zawiera następujące narzędzia do analizy, o których bardziej szczegółową informację można znaleźć w [12].

- *View Kernel* służy do wizualizacji, obejmującej diagram space-time i diagram Gantta. Możliwe jest skorelowanie elementów tych diagramów z fragmentem kodu źródłowego (*source code call-back*).
- *Statistics Kernel* oszacowuje cały zestaw metryk wydajności na podstawie śladu, np. czas wykonania, blokowania i aktywności względem procedur i względem procesora.
- *Index Kernel* oszacowuje szereg indeksów, których wartości mogą wskazywać na problemy wydajności, związane np. ze złym zrównoważeniem obciążenia, przeciążeniem sieci komunikacyjnej, czasem spędzonym w komunikacji lub nieefektywnym używaniem kanałów komunikacyjnych.
- *Modeling Kernel* pozwala na badanie skalowalności aplikacji w funkcji liczby procesorów i rozmiaru zadania.

Ograniczenia. Instalacja i używanie narzędzia są bardzo skomplikowane. W szczególności dotyczy to narzędzia do instrumentacji i biblioteki funkcji przeznaczonych do śledzenia.

Podsumowanie. AIMS jest potężnym narzędziem do oceny zachowania się programów równoległych, umożliwiającym personalizowalną i interaktywną wizualizację przebiegu wykonania programu, ale niewygodny w użyciu.

4.1.4. Medea

Narzędzie Medea zostało opracowane w połowie lat 90. na Uniwersytecie w Pavia (Włochy) [14].

Charakterystyka. Medea jest narzędziem do analizy wydajności programów równoległych typu *off-line*, które pozwala na analizę danych wydajności w oparciu o obliczenie metryk i statystyk. Może wczytywać pliki w formatach PARMON [15], PICL [10], VFCS[16], i ALOG [17], które są przetwarzane na format wewnętrzny narzędzia.

Możliwości. Narzędzie zapewnia możliwości filtrowania i klasteryzacji danych według stopnia detalizacji: na poziomie procesu, procedury, pętli, fragmentu kodu i instrukcji. Generowany jest szereg ważniejszych metryk: profile, przyspieszenie, efektywność i sygnatury. Statystyki obejmują średnią, minimum, maksimum, odchylenie standardowe, korelacje, dystansy składowych, dystrybucję. Analiza klastrowa identyfikuje jednorodne grupy składowych w zależności od wybranych parametrów badania. Wybór ten jest podyktowany obliczeniami macierzy korelacyjnej, która identyfikuje wysoce skorelowane parametry. Ważnym elementem narzędzia jest moduł dopasowywania, który umożliwia opisanie zależności pomiędzy poszczególnymi wskaźnikami jakości działania aplikacji za pomocą wielomianów, funkcji trygonometrycznych, wykładniczych.

Ograniczenia. Dużą niedogodnością jest to, że ślad wykonania programu musi być zapisany w jednym z wymienionych formatów, co w niektórych przypadkach będzie wymagało napisania programu do konwersji śladu z formatu oryginalnego na np. format PICL. Nie można uzyskać postaci źródłowej oprogramowania.

Podsumowanie. Medea jest narzędziem pożytecznym dla analiz statystyk w badaniach wydajności, jednak nie umożliwia badań dynamiki zachowania aplikacji, które są istotne w systemach heterogenicznych.

4.1.5. VAMPIR

VAMPIR [18] został opracowany w połowie lat 90. w ośrodku badawczym KFA Juelich (Niemcy) i później został przejęty przez firmę PALLAS do komercyjnego rozpowszechniania.

Charakterystyka. VAMPIR jest elastycznym i potężnym narzędziem do monitorowania i wizualizacji jakości działania programów równoległych używających bibliotek PVM i MPI. Jest to narzędzie typu off–line. Umożliwia tworzenie i użytkowanie plików śladu na cały szereg sposobów. Pliki śladu są generowane przez moduły VAMPIRtrace, VAMPIRprep i TraceGenerator. Pliki śladu mogą być użyte w formacie tekstowym lub binarnym. Narzędzie pozwala na dynamiczną dekompresję skompresowanych plików śladu. VAMPIR umożliwia użytkownikowi wgląd w wykorzystanie procesora w skali globalnej i lokalnej w postaci wykresów przedstawiających stany programu i procesów. Paleta kolorów może być dopasowana do stanów procesów. Narzędzie może funkcjonować na wielu platformach stacji roboczych.

Możliwości. Narzędzie zapewnia automatyczną instrumentację kodu źródłowego aplikacji, a także ręczne uzupełnienie pliku śladu o dodatkowe znaczniki. Generowanie śladu zdarzeń wprowadza akceptowalny poziom zakłóceń. Istotną zaletą narzędzia jest bardzo dobry interfejs graficzny i system wizualizacji, zapewniający wysoki stopień skalowalności przedstawianej informacji. Istnieje możliwość dekompresji, „w locie" plików skompresowanych za pomocą programów compress i gzip. Narzędzie umożliwia wybór globalnej lub lokalnej wizualizacji poprzez wybór procesu z globalnego widoku. Umożliwione jest powiększanie lub zmniejszanie rozdzielczości przedstawianej informacji w bardzo szerokim zakresie. Dokumentacja jest bardzo dobrze opracowana.

Ograniczenia. VAMPIR nie zapewnia usuwania zakłóceń związanych z monitorowaniem. Nie jest możliwe skorelowanie wyników analizy jakości działania z kodem źródłowym aplikacji. Umożliwione jest podtrzymywanie abstrakcji tylko na poziomie programu lub procesów.

Podsumowanie. VAMPIR zapewnia wysokiej jakości wizualizację zachowania aplikacji, udostępniając potężne możliwości dekompresji pliku śladu i zmiany rozdzielczości przedstawianych informacji. Brak możliwości skorelowania informacji graficznej z kodem aplikacji, niewystarczający zakres poziomów abstrakcji danych o jakości działania aplikacji zmniejszają atrakcyjność tego narzędzia.

4.1.6. CARNIVAL

Carnival został opracowany w latach 1993–1997 w University of Rochester (USA) [25].

Charakterystyka. Carnival jest narzędziem do przeprowadzenia pomiarów i analizy jakości działania programów równoległych w oparciu o algorytm wyszukiwania. Podtrzymuje

hierarchiczną abstrakcję w prezentacji danych wydajności, ustala korelacje między pomiarami dynamicznymi a kodem źródłowym, i umożliwia automatyzację analizy przyczynowo-wynikowej zjawisk wydajności. Implementacja narzędzia jest przeznaczona dla aplikacji, funkcjonujących na maszynach z pamięcią rozproszoną z przesyłaniem komunikatów. W celu instrumentacji kodu Carnival wykonuje dwustopniową kompilację. Przy wykonaniu kod obiektowy aplikacji korzysta z programu do rejestracji śladu UTE [26]. Celem instrumentacji jest wyeksponowanie semantycznej struktury aplikacji i szczegółów zrównoleglenia niejawnie obecnych w oryginalnym kodzie. Przy opracowaniu narzędzia zostało wykorzystane środowisko Sage++ [27].

Implementacja narzędzia funkcjonuje na platformach IBM SP2 i SGI dla programów w HPF.

Możliwości. Carnival wykonuje pomiary i przedstawia profile wydajności dla kilku kategorii czasu wykonania takich, jak lokalne obliczenia, obliczenia równoległe, narzut zrównoleglenia (np. pakowanie i rozpakowanie danych), operacje odbioru i wysyłania, synchronizacja, oczekiwanie, narzut wielozadaniowości.

Najważniejszym aspektem i zaletą tego narzędzia jest mechanizm wnioskowania o przyczynach powstawania czasu przestoju, oparty na metodzie porównywania śladów poszczególnych procesów przy wykonaniu programu i wykrywaniu zależności między nimi. Metoda ta służy do istotnego zmniejszenia nieistotnej informacji w śladzie wykonania poprzez wygenerowanie klas ekwiwalencji kroków i odrzucanie kroków wykonania, powtarzających się w różnych procesach. W wyniku filtracji w śladzie pozostają tylko te zdarzenia wykonania, które powodują przestoje procesów. Skorelowanie danych analizy z kodem źródłowym pozwala na identyfikację tych fragmentów kodu, które powodują niepożądane zjawiska straty czasu.

Ograniczenia. Zastosowanie narzędzia jest możliwe tylko pod warunkiem homogeniczności węzłów. Inną poważną niedogodnością jest prawie nie sterowalne generowanie przez program śledzący ogromnej ilości danych dochodzącej do kilkuset MB. Ręczna instrumentacja kodu aplikacji umożliwia generację umiarkowanej ilości danych, ale jest pracochłonna.

Podsumowanie. Carnival na podstawie porównania śladów wykonania poszczególnych procesów umożliwia automatyzację procesu wnioskowania przyczynowo-wynikowego i wykrywa w kodzie aplikacji te decyzje implementacyjne, które powodują powstawanie przyczyn strat wydajności. Nie można go wykorzystywać w systemach heterogenicznych.

4.2. Narzędzia do analizy i wizualizacji w trybie on–line

4.2.1. XPVM

Narzędzie XPVM powstało w Oak Ridge National Laboratory w połowie lat 90. jako sukcesor narzędzia Xab [19].

Charakterystyka. XPVM jest programem–konsolą do monitorowania i wizualizacji programów równoległych, którego oryginalność polega na tym, że wychwytuje każde zdarzenie, wygenerowane przez aplikacje i odwzorowuje je natychmiast na widokach. Format danych w XPVM jest samoopisujący i dlatego jest łatwo konwertowany na inne formaty, przyjęte w modelu programowania PVM. Jednak XPVM nie używa samoopisującej części formatu i realizuje bezpośrednie odczytywanie zdarzeń dzięki znajomości składni i semantyki.

Możliwości. XPVM zapewnia pięć widoków. Najważniejszymi są animacja aktywności wirtualnej maszyny równoległej, na której funkcjonuje aplikacja i diagram space-time. Jakość wizualizacji jest o wiele wyższa, niż ta oferowana przez ParaGraph, ponadto diagram space-time jest interaktywny, czyli jest możliwe kliknięcie na przedstawiane obiekty i uzyskanie informacji o odpowiednim zdarzeniu. Ponadto XPVM przedstawia przebieg wywołań funkcji komunikacyjnych w postaci tekstowej, co umożliwia debugowanie programu w ograniczonym zakresie.

Ograniczenia. Wizualizacja w XPVM nie jest personalizowalna, oraz nie daje się rozszerzyć. Widoki XPVM są mało dokładne ze względu na bezpośrednią rejestrację zdarzeń: zakłócenia wykonania są bardzo znaczące i poza wychwytywaniem zdarzeń dużo czasu zajmuje przesyłanie danych w wirtualnym komputerze, realizowane obok właściwej komunikacji w aplikacji.

Podsumowanie. W założeniu XPVM miał zostać prostym w użyciu narzędziem do wizualizacji i ograniczonego debugowania, ale do oceny jakości programu jest mało przydatny ze względu na zbyt małą dokładność przedstawianych danych.

4.2.2. JEWEL

System JEWEL powstał na przełomie lat 80. i 90. w ośrodku badawczym GMD w Sankt Augustin (Niemcy) [20].

Charakterystyka. JEWEL jest przedstawicielem hybrydowego podejścia do instrumentacji i budowy mechanizmu zbierania danych o aplikacjach równoległych. System składa się z podsystemów: podsystemu zbierania i redukcji danych, podsystemu graficznej prezentacji oraz podsystemu sterowania eksperymentem. W systemie są wprowadzone pojęcia wskaźnika wydajności, aspektu i poziomu szczegółowości. Wskaźniki wydajności są zdefiniowane jako wielkości, charakteryzujące wydajność badanego systemu, czyli wielkości surowe, natomiast pojęcie aspektu jest pojęciem syntetycznym służącym charakteryzacji określonego elementu systemu lub grupy elementów (np. komunikacja w określonych procesach programu). Aspektowi odpowiada zbiór wskaźników wydajności. Badany program jest logicznie przedstawiany jako zbiór aspektów. Najwyższy poziom szczegółowości obejmuje cały zbiór wskaźników wydajności, zaś niższy definiuje tylko podzbiór wyższego poziomu.

Możliwości. Na podstawie zbioru aspektów można stworzyć konfigurację eksperymentu. Badanie poszczególnych aspektów może być dynamicznie włączane i wyłączane. Podsystem sterowania eksperymentem jest rozproszony. System umożliwia monitorowanie wykonania programów w trybie *on-line* oraz rejestrację przebiegu zachodzenia interesujących użytkownika zdarzeń do dalszej analizy *off-line*. Konfigurowanie aspektów wydajności, sterowanie eksperymentem i późniejsza analiza danych wydajności są prowadzone za pomocą bardzo wygodnego interfejsu graficznego. Czas globalny w systemie zapewnia się za pomocą kart VME-bus, co pozwala na osiągniecie bardzo małych rozpływów czasów na węzłach rzędu 0,2 mikrosekundy. Zakłócenia wprowadzane przez instrumentację są rzędu kilkudziesięciu mikrosekund.

Ograniczenia. System umożliwia instrumentację obiektów w badanym programie wyłącznie w ręczny sposób. Istotnym elementem środowiska, różniącym je od innych narzędzi tego typu jest dedykowana lokalna sieć pomiarowa, której stosowanie znacznie zmniejsza poziom zakłóceń w badanym systemie. JEWEL funkcjonuje w systemach operacyjnych Amoeba i Mach.

Podsumowanie. Pod względem konfigurowalności sterowania eksperymentu i prezentacji jakości działania badanego programu JEWEL jest bardzo dogodny. Zależność funkcjonowania systemu od wspomnianych rozwiązań sprzętowych ogranicza jego stosowalność na szerszą skalę.

4.3. Narzędzia do konfigurowalnej analizy i wizualizacji

4.3.1. Pablo

Zestaw narzędzi Pablo (*Pablo toolkit*) powstał w latach 1988–1994 w University of Illinois w Urbana–Champaign [21].

Charakterystyka. Pablo opracowano do przetwarzania śladów w sposób ogólny (*generic*) dla zapewnienia rozbudowywalności. Pablo nie jest tylko narzędziem do wizualizacji – jego celem jest kontrola całego procesu oceny jakości działania, instrumentacji kodu i wizualizacji jego wykonania. Pablo składa się z biblioteki funkcji śledzących, graficznego edytora do konfigurowania grafów analizy i wizualizacji, szeregu modułów do filtrowania i wizualizacji danych oraz dużego zestawu programów pomocniczych. Ślady, używane przez Pablo są zapisywane w formacie samoopisującym SDDF [22].

Możliwości. Pablo jest pierwszym narzędziem do wizualizacji wydajności, które zostało wyposażone w wizualny język programowania. Język ten pozwala na konstruowanie grafów analizy i wizualizacji z połączeniem składników, przy czym składnikami mogą być filtry do analizy danych lub wizualizacji. Dane przepływają od jednego składnika do drugiego, przy czym każdy składnik przed przekazaniem danych do następnego w łańcuchu składnika przetwarza je. Jednocześnie możliwe jest rozszerzanie grafu przetwarzania, wystarczy stworzyć nowy składnik i dołączyć go do grafu. Pablo jest wyposażony w dosyć szeroką gamę takich składników. Moduł programowania wizualnego używa SDDF również do przetwarzania danych, ponieważ dane zapisywane są w tym formacie, oraz dlatego że pozwala on na realizację polimorficznych składników (np. liczby są zapisywane w postaci liczb całkowitych lub rzeczywistych oraz dokonuje się niezbędnych konwersji).

Ograniczenia. Wizualizacje w Pablo są wyłącznie statystyczne i nie są interaktywne. Pablo jest bardzo trudno używać bez praktyki. Konfigurowanie schematu analizy programowania wizualnego jest zadaniem trudnym i nieco odstrasza od posługiwania się nim; dotkliwy jest brak modułów logicznych. Pablo jest włączony do szeregu eksperymentalnych wizualizacji, albo wielowymiarowych, albo posługujących się środkami rzeczywistości wirtualnej; jednak te wizualizacje są trudno eksploatowalne.

Wersja Pablo 5.0. W Pablo w wersji 5.0 usunięto wspomniane moduły graficznej prezentacji, a rozbudowano o jednolity interfejs graficzny, umożliwiający cały cykl działań, począwszy od instrumentacji po analizę, przy czym analiza została sprowadzona do prezentacji tabularnej danych statystycznych. Narzędzie można stosować do analizy aplikacji napisanych w języku HPF i C, w tym z użyciem MPI.

Podsumowanie. Pomysł stworzenia narzędzia funkcjonalnie niezależnego od semantyki aplikacji można ocenić jako bardzo dobry, praktyka zaś wyjawiła bardzo znaczący koszt implementacyjny i użytkowy tego projektu. Format SDDF, który powstał w ramach prac nad Pablo, jest zaakceptowany przez konsorcjum Ptools jako standard dla narzędzi tego typu.

4.3.2. SCOPE

Narzędzie do wizualizacji SCOPE[6] zostało opracowane w pierwszej połowie lat 90. w Institut National Polytechnique de Grenoble (Francja).

Charakterystyka. SCOPE jest narzędziem do wizualizacji typu *off-line*, opartym na odtwarzaniu śladu wykonania programu, bardzo rozbudowanym pod względem możliwości wizualizacyjnych i graficznych. Oferuje model programowania wizualnego, który umożliwia stworzenie wizualnego grafu analizy i prezentacji. Graf ten składa się z połączonych z sobą składowych:

- zależnych od semantyki śladu, mogą być nimi czytniki śladu, symulatory, generujące stan systemu oraz prezentery,
- składowe ogólne (*generic*), np. filtry statystyczne,
- moduły do prezentacji informacji w postaci wykresów,
- moduły do sterowania systemem.

Możliwości. Wspomniane składowe specyficzne umożliwiają graficzny dobór danych oraz sposobu ich przetwarzania, niezbędnego do przedstawienia poszczególnych zjawisk działania. Narzędzie zapewnia bogatą gamę wizualizacji, na którą składają się 2- i 3-wymiarowe prezentacje statystyczne i animacyjne, każdą z nich można dostrajać pod względem specyfiki danych zawartych w śladzie. Skalowalność narzędzia jest zapewniona za pomocą mechanizmów filtracji i klasteryzacji danych.

Ograniczenia. Narzędzie nie jest dostępne dla zewnętrznych użytkowników.

Podsumowanie. Z punktu widzenia wymogów co do możliwości narzędzie to umożliwia personalizowalną, interaktywną i rozszerzalną wizualizację przebiegu wykonania programów. Obecnie narzędzie jest przeznaczone dla programów, które używają biblioteki PVM. Kolejna edycja narzędzia ma być przeznaczona do funkcjonowania w środowisku rozproszonym Athapascan, opartym o model programowania „klient – serwer".

4.3.3. Tau – pC++

Zestaw narzędzi Tau (Tuning and Analysis Utilities) powstał w połowie lat 90. w University of Oregon w Eugene (USA) [32].

Charakterystyka. Tau jest zestawem narzędzi do analizy wydajności programów równoległych dedykowanych na obiektowy system programowania równoległego pC++. Założeniem modelu środowiska jest integracja wymogów do analizy programu równoległego: analiza językowo-semantyczna aplikacji, przenośna infrastruktura pomiarowo-analityczna oraz analiza zintegrowana z kompilatorem. Podstawowym założeniem języka pC++ jest pojęcie rozproszonego zbioru. pC++ zapewnia prosty mechanizm tworzenia obiektów z bazowej klasy element. Funkcje z klasy element mogą być stosowane do całego zbioru lub podzbioru równocześnie. Ten mechanizm zapewnia użytkownikowi jasny interfejs z operacjami typu data-parallel za pomocą wywołania funkcji członkowskich klasy bazowej.

Możliwości. Dedykowanie środowiska Tau na język pC++ implikuje specjalne wymagania do narzędzi, które polegają na:

- zapewnieniu wglądu w zachowanie obiektów typu zbiór, klasa, metoda i funkcja,
- integracji z kompilatorem i systemem wykonania pC++ dla celów instrumentacji i uzyskania dostępu do właściwości obiektów programu,

- zapewnieniu przenośności interfejsu graficznego narzędzia,
- implementacji wysokiego poziomu przyjazności dla użytkownika, m.in. poprzez wbudowanie mechanizmu podobnego do odnośników w systemach hipertekstowych.

Rdzeniem wszystkich działań nad aplikacją jest tworzone przez kompilator abstrakcyjne drzewo syntaktyczne (AST), do którego dostęp jest możliwy za pomocą biblioteki Sage++ [27]. Na zestaw narzędzi do analizy statycznej składają się programy do globalnego przeglądania funkcji i metod *fancy*, prezentacji grafu wywołań *cagey* i hierarchii klas *classy*. W zestawie narzędzi do analizy dynamicznej:

- *racy* zapewnia przeglądanie profilów,
- *easy* służy do śledzenia sekwencji abstrakcyjnych zdarzeń,
- *breezy* umożliwia debugowanie wykonania programu.

System 3–D wizualizacji dynamicznej zapewnia wysoką jakość animacyjnej prezentacji przebiegu wykonania aplikacji.

Ograniczenia. Narzędzie jest przydatne do animacji zachowania aplikacji, ale nie zapewnia tradycyjnych wizualizacji. Rozszerzenie zaś narzędzi nie jest możliwe ze względu na brak dostępu do kodu źródłowego.

Podsumowanie. Wykorzystanie wiedzy o właściwościach docelowego języka programowania umożliwia uzyskanie głębszego wglądu w zachowanie programu za pomocą systemu kompilującego.

4.4. Narzędzia z instrumentacją dynamiczną

4.4.1. Annai

System narzędziowy Annai [23] opracowany został w Centro Svizzero di Calcolo Scientifico (CSCS), które jest filią Eidgenossische Technische Hochschule (ETH) w Manno (Szwajcaria).

Charakterystyka. Annai obejmuje 3 narzędzia programowe, dzielące jeden interfejs graficzny: narzędzie do wspomagania zrównoleglenia (PST), narzędzie do równoległego debugowania i narzędzie do pomiarów i analizy wydajności (PMA). Narzędzia Annai wykonują swoje funkcje w trybie *on-line*. Annai użyteczne dla programów napisanych w HPF, jak również używających biblioteki MPI. Narzędzie PST odgrywa rolę kompilatora dla tych dwóch typów programów.

Możliwości. Annai/PMA służy do instrumentacji biblioteki komunikacyjnej, systemu kompilacji i instrumentacji dynamicznej. Kod instrumentacji biblioteki komunikacyjnej i systemu kompilacji domyślnie jest nieaktywny. Przy uruchomieniu programu użytkownik może skonfigurować kod instrumentacji, np. do generacji profilu wykonania procedur.

Program śledzący Annai może funkcjonować w trybie profile lub w trybie śledzenia. Użycie biblioteki zainstrumentowanych funkcji komunikacyjnych do instrumentacji aplikacji nie tylko eliminuje konieczność rekompilacji programu, lecz też umożliwia obserwację komunikacji na niższym poziomie. W ten sposób Annai/PMA pozwala na rozróżnianie stanu blokowania od stanu czytania danych podczas odbioru komunikatu.

Instrumentacja systemu kompilacji pozwala na obserwację programu na różnych poziomach abstrakcji kodu źródłowego: funkcje, zagnieżdżone pętle, bloki warunkowe i bloki instrukcji. Struktura hierarchiczna tych składników jest znana użytkownikowi i może być użyta do przedstawienia metryk w sposób strukturyzowany i łatwo zrozumiały.

Oryginalność Annai/PMA polega na tym, że pozwala ono na zmianę konfiguracji w sposób dynamiczny w trakcie wykonania programu. Można wyspecyfikować punkty działania instrumentacji (*instrumentation action points*), na które napotkał przebieg wykonania.

Twórcy Annai/PMA szczególną uwagę poświęcili ocenie perturbacji, wprowadzanej przez rejestrację informacji wydajności [23]. Na wizualizacjach graficznych, perturbacje są przedstawione w postaci specjalnego stanu, co pozwala na lokalizację miejsc w programie, które są najbardziej zakłócone. Do oszacowania śladów Annai oferuje konfigurowalne narzędzie do wydajnej 2- i 3-wymiarowej wizualizacji.

Ograniczenia. Annai ma zastosowanie ograniczone do NEC SX–4. Annai nie oblicza przybliżenia dynamiki niezainstrumentowanego programu, a tylko stwierdza fakt i oszacowuje stopień perturbacji.

Podsumowanie. Zastosowanie w Annai kilku typów instrumentacji umożliwia dynamicznie sterowalne zbieranie danych oraz eliminuje konieczność rekompilacji aplikacji.

4.4.2. Paradyn

Narzędzie Paradyn zostało opracowane w połowie lat 90. w University of Wisconsin w Madison (USA).

Charakterystyka. Paradyn dokonuje instrumentacji dynamicznej programu w trakcie wykonania. Instrumentacja jest sterowana przez użytkownika. Badanie problemu wydajności jest zaimplementowane za pomocą modułu *performance consultant*.

Możliwości. W odróżnieniu od innych narzędzi, w których są generowane zdarzenia odpowiadające zmianom stanów aplikacji, Paradyn pobiera co jakiś czas próbki wskaźników jakości określonych przez użytkownika. Ślady są mniejsze niż ślady klasyczne, jednak nie umożliwiają odtwarzania stanów aplikacji.

Sesja oceny jakości działania jest sterowana za pomocą okna zawierającego prezentację modułów badanej aplikacji, która pokazuje jaka część aplikacji jest w danym momencie wykonywana.

Badanie problemu wydajności za pomocą modułu *performance consultant* obejmuje tworzenie modelu do charakteryzacji problemów wydajności w 3 osiach (*gdzie, dlaczego, kiedy*), które odpowiadają lokalizacji problemu w aplikacji (*gdzie*), w sekwencji działań (*dlaczego*) i w czasie (*kiedy*).

Kiedy wskaźnik wydajności wychwytuje anomalną wartość, moduł szuka punktu, odpowiadającego problemowi w przestrzeni przeszukiwań W^3, a następnie przedstawia graficzną ewolucję wskaźnika wydajności, który wykazał anomalię.

Paradyn oferuje 2 wizualizacje interaktywne, ślad histogramu i wizualizację macierzy wskaźników. Narzędzie jest wyposażone w programowalny interfejs, umożliwiający wizualizacje zewnętrzne z przedstawianiem wskaźników generowanych w trakcie wykonania.

Ograniczenia. Główną niedogodnością Paradynu jest brak wizualizacji klasycznych, np. diagramu *space-time.* Brak przenośności jest związany z instrumentacją dynamiczną kodu, co wymaga dostępu do przestrzeni adresowej aplikacji. Taki dostęp nie jest możliwy we wszystkich systemach lub może wymagać realizacji funkcji specyficznych na bardzo niskim poziomie.

Podsumowanie. *Performance consultant* Paradynu jest prototypem narzędzia przyszłości. Badanie wydajności staje się coraz bardziej pracochłonne. Dostępność tego typu narzędzi

powinno uwolnić użytkownika od obowiązku obserwacji przebiegu wykonania programu. Z drugiej strony, ze względu na brak adekwatnych śladów, Paradyn nie może zastąpić narzędzi tradycyjnych, które dostarczają danych, opartych o sekwencję zdarzeń aplikacji.

4.5. Narzędzia o standaryzowanych interfejsach

4.5.1. SIMPLE

Zestaw narzędzi SIMPLE powstał na przełomie lat 80. i 90. na Uniwersytecie Erlangen–Nuernberg (Niemcy) [28].

Charakterystyka. SIMPLE jest przedstawicielem „standaryzującego nurtu" w dziedzinie analizy jakości działania programów równoległych, którego założeniem jest jednolity interfejs dostępu do danych i operacji nad nimi, zdefiniowany za pomocą specjalnego języka do opisywania śladu zdarzeń. W założeniu miało to być narzędzie do analizy śladów zdarzeń, pochodzących z monitorowania wykonania programów na sieci PC przez system sprzętowy ZM4. Zbieranie danych może być przeprowadzone w sposób sprzętowy lub hybrydowy (programowo-sprzętowy) za pomocą instrumentacji oprogramowania badanego systemu.

Możliwości. SIMPLE ma modularną strukturę i standaryzowane interfejsy, co ma ułatwić rozbudowę istniejących narzędzi i dołączenie nowych. Niezależność narzędzi, składających się na SIMPLE od badanych obiektów oparta jest na dwóch zasadach. *Pierwszą zasadą* jest to, że wszystkie narzędzia używają interfejsu TDL/POET/FILTER do dostępu do danych śladu. Zanim zostanie wczytany plik śladu, wczytywany jest opis formatu w języku TDL. Ma to kilka zalet: narzędzia są niezależne od formatu śladu; wszystkie narzędzia są dopasowywane do nowego formatu śladu za pomocą napisania tylko jednego opisu TDL (Trace Definition Language); wszystkie narzędzia mają ten sam interfejs użytkownika, który zapewnia filtrację rekordów zdarzeń. Mając stworzony opis filtru, użytkownik może stosować go we wszystkich narzędziach, z których składa się SIMPLE. *Drugą zasadą* jest niezależność narzędzi od semantyki danych śladu. Każde narzędzie zapewnia swój język komend i konfiguracji. Skonfigurowany plik komend narzędzia może być użyty powtórnie dla innych aplikacji.

Ograniczenia. Rozpowszechnieniu tego środowiska przeszkodził szereg restrykcji licencyjnych oraz brak dostępu do kodu źródłowego, ponieważ szereg architektur stacji roboczych został pominięty przez autorów SIMPLE.

Podsumowanie. Zasada uniezależnienia dostępu narzędzi do danych od formatu i treści pliku śladu jest bardzo ważna przy budowie narzędzi. Z drugiej strony, zapoznanie się z regułami środowiska może zająć dużo czasu, dlatego uważa się, że jest to środowisko przeznaczone bardziej dla ekspertów, niż dla zwykłych użytkowników. Technika definiowania interfejsu dostępu do danych za pomocą TDL miała być konkurencyjna w stosunku do formatu SDDF, zaś praktyka pokazała przewagę tego ostatniego.

4.5.2. TOOLSET

Zestaw narzędzi TOOLSET [29] powstał w Technische Universitaet Muenchen (Niemcy), będąc kontynuacją wcześniejszych prac nad środowiskiem TOPSYS [30].

Charakterystyka. TOOLSET obejmuje narzędzia typu *on-line* do debugowania, równoważenia obciążenia, analizy wydajności, podtrzymywania równoległego I/O i wizualizacji.

Możliwości. TOOLSET składa się z następujących narzędzi:

- **PATOP** służy do analizy jakości działania aplikacji. Ma wersję o nazwie TATOO, która funkcjonuje w trybie *off-line*,
- **VISTOP** jest narzędziem do animacyjnej prezentacji przebiegu wykonania aplikacji,
- **DETOP** służy do debugowania aplikacji,
- **LoMan** jest przeznaczone do sterowania równoważeniem obciążenia,
- **CoCheck** jest narzędziem do prowadzenia *checkpointingu*.

W chwili obecnej zestaw jest w przebudowie ze względu na wprowadzenie systemu monitorującego OCM o dobrze zdefiniowanym interfejsie z pozostałymi narzędziami jako warstwy dodatkowej w strukturze przetwarzania danych o jakości działania zgodnej z założeniami standardu OMIS [31]. Projekt ten powinien umożliwić budowę zintegrowanego zestawu narzędzi, które będą wspierać tworzenie programów równoległych od samego początku testowania programu aż do oddania go do użytku. Danych ma dostarczać bardzo elastyczny mechanizm monitorowania. Przewidywana jest adaptacja systemu monitorującego do współpracy ze standardem MPI, a w dalszej przyszłości na architekturach typu DSM.

Ograniczenia. Uniwersalność systemu monitorującego jest związana z dużym nakładem pracy, dlatego pod znakiem zapytania pozostaje kwestia dojrzałości implementacyjnej systemu.

Podsumowanie. Niewątpliwą zaletą projektu TOOLSET jest założenie, że wszystkie narzędzia będą oparte o standard współpracy z jednolitym systemem monitorującym, co pozwoli uniezależnić funkcjonowanie narzędzi od platformy, systemu operacyjnego, biblioteki komunikacyjnej i wprowadzić mechanizm programowalności świadczenia usług ze strony systemu monitorującego.

5. Porównanie i ocena przydatności narzędzi

Można zauważyć, że istnieje przewaga liczbowa narzędzi typu *off-line* nad narzędziami typu *on-line*. Przyczyną tego są trudności implementacji systemu monitorującego, który jest dostawcą danych o wydajności. Stosunkowo mało narzędzi pozwala na interaktywną instrumentację kodu aplikacji. Większość narzędzi jest zależna od środowiska programowania równoległego, co czyni ich przenośność ograniczoną.

W tabeli 2 przedstawione są oceny jakości narzędzi w funkcji powyżej przedstawionych kryteriów, opracowane przez autora na podstawie ocen jakościowych podanych w cytowanej literaturze, w 10-punktowej skali. Symbol „–" oznacza ocenę zerową.

Środowiska bardziej skomplikowane nie koniecznie są oceniane najlepiej: można powiedzieć, że narzędzie proste może oferować lepsze możliwości związane z wizualizacją różnych typów śladu podczas, gdy interakcja użytkownika z systemem wizualizacyjnym w narzędziu ze skomplikowaną manipulacją czyni posługiwanie się nim niewygodnym.

Ustalenie jedynej oceny dla jakości narzędzi nie jest możliwe ze względu na wieloaspektowośc ich funkcjonowania i dlatego przy doborze narzędzia należy posługiwać się całym zestawem charakterystyk.

Tabela 2

Tabela porównawcza ocen narzędzi wg skali 10–punktowej

Name	Ext	Gen	Prs	Ita	Pwr	Usr	Iop	Htg	Prt	Acc	Lvl	Mtr
AIMS	6	7	4	6	7	5	5	7	6	7	7	5
Annai	–	2	5	6	7	7	4	–	–	5	7	6
Carnival	–	5	3	6	6	5	3	–	2	6	7	5
JEWEL	4	4	6	8	7	6	5	–	2	9	6	6
Medea	–	6	5	7	5	3	4	4	5	7	6	4
Pablo	7	9	–	–	7	2	3	5	6	5	6	3
Paradyn	5	5	–	–	7	2	3	5	6	5	6	3
ParaGraph	2	–	2	–	5	4	–	6	9	2	3	8
Scope	7	7	9	6	6	7	–	4	5	5	7	4
SIMPLE	6	7	7	2	6	2	5	3	5	6	5	3
Tau	–	–	5	8	7	6	5	–	6	5	7	4
TOOLSET	7	6	6	8	7	5	8	8	6	7	8	5
Upshot	–	9	8	7	4	5	4	6	5	5	4	7
VAMPIR	3	5	6	6	6	7	5	–	6	6	4	6
XPVM	–	–	–	3	4	5	4	8	8	–	4	6

Acc	Dokładność	Lvl	Poziomy abstrakcji
Ext	Rozbudowywalność	Mtr	Dojrzałość
Gen	Ogólność	Prs	Personalizacja
Htg	Heterogeniczność	Prt	Przenośność
Iop	Interoperabilność	Pwr	Moc
Ita	Interaktywność	Usr	Przyjazność dla użytkownika

6. Kierunki rozwoju narzędzi

W pracy przedstawiono stan obecny dziedziny narzędzi do analizy jakości działania programów równoległych jako osobnej dyscypliny analizy wydajności systemów. Została pokazana złożoność zagadnień, które trzeba rozwiązać opracowując takie narzędzia.

Analiza tendencji rozwoju narzędzi [35, 33, 34, 36, 37] pokazuje, że techniki budowy narzędzi będą rozwijały się w następujących kierunkach:

- instrumentacja programów aplikacyjnych powinna być oparta o analizę kodu źródłowego programu, uwzględniającą różne poziomy abstrakcji,
- analiza jakości działania powinna umożliwiać uzyskanie nowych metryk wykonania programu; powinny być opracowane narzędzia oparte o algorytmy automatyzacji wyszukiwania przyczyn niewystarczającej wydajności,
- wizualizacja jakości działania powinna umożliwiać łatwe dodawanie nowych informatywnych prezentacji dla istniejących i nowych danych jakości działania,
- interfejs graficzny użytkownika narzędzia typu *on-line* powinien być wyposażony w mechanizm sterowania procesem monitorowania za pomocą języka usług, których żądania mogłyby być przetwarzane w sposób rozproszony,

- powinna być umożliwiona interoperabilność narzędzi służących dostrajaniu wydajności: do debugowania, analizy jakości działania, wizualizacji, równoważenia obciążenia, równoległego I/O, checkpointing-u itd.
- system monitorowania powinien funkcjonować niezależnie od pozostałych narzędzi oraz powinien być postrzegany jako system świadczący usługi narzędziom za pomocą dobrze zdefiniowanego interfejsu; korzystanie z systemu monitorującego powinno być programowalne.

7. Koncepcja narzędzia PARNAS

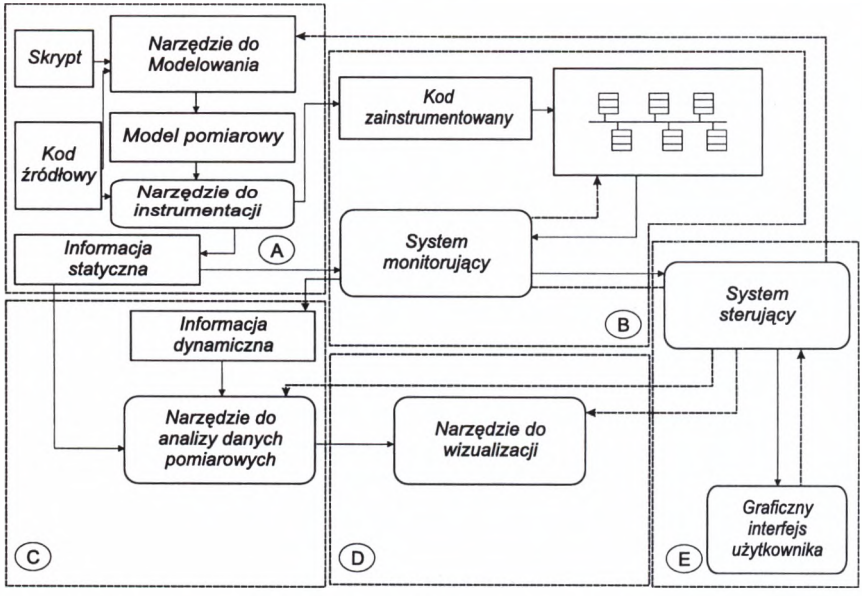
Ograniczenia koncepcyjne, implementacyjne i użytkowe przedstawionych powyżej narzędzi sprawiły, że powstał projekt narzędzia do analizy jakości działania programów równoległych PARNAS, który miał na celu zintegrowanie najlepszych technik i elementów istniejących narzędzi. Implementacją projektu jest prototyp narzędzia typu *off-line* ze sterowalnym monitorowaniem aplikacji.

Narzędzie pozwala na synchronizację zegarów lokalnych i usuwanie zakłóceń wprowadzanych przez monitorowanie.

Instrumentacja jest wykonywana na kodzie źródłowym i może być przeprowadzona automatycznie na podstawie skryptu lub interaktywnie za pomocą graficznego interfejsu użytkownika.

Narzędzie umożliwia szereg wizualizacji statystycznych i dynamicznych, m.in. profile czasu wykonania, diagram Gantta, diagramy aktywności komunikacyjnej aplikacji.

Na rys. 2 przedstawiona jest struktura narzędzia.



Rys. 2. Struktura narzędzia PARNAS: A – instrumentacja, B – eksperyment pomiarowy, C – analiza danych, D – wizualizacja, E – sterowanie

Narzędzie zastosowano do analizy kilku aplikacji równoległych, używających biblioteki PVM, co pozwoliło na określenie zależności strat wydajności od poszczególnych czynników algorytmu i warunków wykonania. Szczegóły dotyczące funkcjonowania narzędzia można znaleźć w [38, 39, 40].

8. Podsumowanie

Przedstawione w pracy narzędzia obejmują szeroki wachlarz możliwości, udostępnianych programistom do uzyskania wglądu w zachowanie programów aplikacyjnych w celu znalezienia tak zwanych wąskich gardeł wykonania i ewentualnej modyfikacji algorytmu. Narzędzia te mogą być stosunkowo proste tak pod względem swoich możliwości, jak i pod względem ich użytkowania jak np. Upshot, mogą być również bardzo skomplikowane pod każdym względem, jak np. AIMS lub Paradyn.

Wzrost możliwości narzędzia wiąże się ze skomplikowaniem posługiwania nim, z drugiej zaś strony ze stopnia złożoności narzędzia nie wynika wzrost przydatności narzędzia, czego przykładem może być środowisko Pablo. Przydatność narzędzia w dużym stopniu jest związana z miarą przyjazności dla użytkownika; narzędzia do wizualizacji powinny być łatwe do zrozumienia i zastosowania, w przeciwnym wypadku użytkownicy będą woleli prostsze narzędzia od bardziej skomplikowanych nawet kosztem zmniejszenia możliwości.

Wysoki stopień złożoności implementacji systemu monitorującego sprawia, że istnieje poważny problem niezgodności istniejących narzędzi, opracowanych w różnych ośrodkach badawczych. Dlatego jednym z najważniejszych wyzwań, stojących przed dziedziną budowy narzędzi i środowisk jest zapewnienie interoperabilności narzędzi poprzez standaryzację mechanizmów interakcji pomiędzy systemem monitorującym a narzędziami służącymi właściwej analizie jakości działania programów równoległych. Obiecującym rozwiązaniem tego problemu jest zaproponowany standard OMIS [31].

Prace nad rozbudową możliwości narzędzia PARNAS będą kontynuowane w kierunku uniezależnienia od siebie programowalnego systemu monitorującego i zestawu narzędzi do analizy w oparciu o wspomniany wyżej standard OMIS. Ma to przynieść wymierne korzyści w postaci niezależności narzędzi od platformy sprzętowej.

Literatura

[1] *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for networked Parallel Computing*. Cambridge, Massachusetts, MIT Press 1994

[2] Dongarra J., Hempel R., Hey A.J.G., Walker D.W.: *A proposal for a user-level, message passing interface in a distributed memory environment*. Report ORNL/TM–12231, 1993

[3] Jain R.: *The art of computer systems performance analysis*. John Wiley & Sons 1991

[4] Malony A.D.: *Performance Observability*. PhD Thesis, University of Illinois Urbana-Champain, 1990

[5] Hollingsworth J.K., Lumpp J.E., Miller B.P.: *Techniques for performance measurement of parallel programs.* Parallel Computers: Theory and Practice (IEEE Press), 1995, 225–240

[6] Arrouye Y.: *Environments de visualization pour l'evaluation des performances des systemes paralleles: etude, conception et realisation.* These de PhD, Institut National Polytechnique de Grenoble, november 1995

[7] http://www.ptools.org

[8] http://www.nhse.org

[9] Heath M.T., Finger J.E.: *ParaGraph: A Tool for Vizualizing Performance of Parallel Programs*. Oak Ridge National Laboratory, 1994; http://www.netlib.org/paragraph/

[10] Geist G.A., Heath M.T., Peyton B.W., Worley P.H.: *A User's Guide to PICL: a portable instrumented communications library*. Technical Report No. ORNL/TM–11616, Oak Ridge, Tennessee, Oak Ridge National Laboratory, January 1992

[11] Lusk E.: *Upshot*, Technical Report CIS–TR–92–21, Argonne National Laboratory, ANL 97403, October 1992, http://www–c.mcs.anl.gov/home/lusk/upshot/index.html; http://www.mcs.anl.gov/mpi/mpich/

[12] Yan J.C.: *Performance tuning with AIMS – an automated instrumentation and monitoring system for multicomputers,* [w:] *Proceedings of the 27th Hawaii International Conference on System Sciences, Hawaii,* 1994; http://science.nas.nasa.gov/Software/Archives/

[13] Sarukkai S.R., Malony A.D.: *Perturbation analysis of high level instrumentation for SPMD programs,* [w:] ACM SIGPLAN San Diego, CA, July 1993, Notices 44–53

[14] Calzarossa M., Massari L., Merlo J., Tessera D.: *Parallel Performance Evaluation: The Medea Tool,* [w:] *Liddell, H., Colbrook, A., Hertzberger, B., Sloot, P., (eds.), Proc. Int. Conf. High,, Performance Computing and Networking*, Brussels, Belgium, April 1996, Springer–Verlag Lecture Notes in Computer Science 1067, 1996, 522–529; e–mail: medea@gilda.unipv.it

[15] Pozetti E., Serazzi G., Vetland V.: *ParMon – A tool for monitoring parallel programs*, Technical Report CNR *Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo*, R3/148, Rome, 1994.

[16] Zima H.P., Chapman B.: *Compiling for Distributed–Memory Systems*, [w:] *Proc. of the IEEE*, 81(2), 1993, 264–287,

[17] Butler R., Lusk E.: *Monitors, Messages, and Clusters: The p4 Parallel Programming System*, [w:] Parallel Computing, 20, 1994, 547–564

[18] *VAMPIR/VAMPIRtrace – Performance Analysis of MPI Programs;* http://www.pallas.de/pages/vampir.htm

[19] Kohl J.A., Geist G.A.: *The PVM 3.4 Tracing facility and XPVM 1.1.* [w:] *Proceedings of the 29th Hawaii International Conference on System Sciences*, Hawaii, 1996; http://www.netlib.org/utk/icl/xpvm/xpvm.html

[20] Lange F., Kroeger R., Gergeleit M.: *JEWEL: Design and Implementation of a Distributed Measurement System*, [w:] *IEEE Transactions on Parallel and Distributed Systems*, 3(6), November 1992, 657–671; http://wsv.gmd.de/aiw/AP.htm\#1992 (Arbeitspapiere Nr 710)

[21] Reed D.A., Aydt R., Madhyastha T.M., Noe R.J., Shields K.A., Schwartz B.W.: *The Pablo Performance Analysis Environment*. Technical Report, Dept. of Comp. Sci., University of Illinois, 1992;
http://www–pablo.cs.uiuc.edu/Projects/HPF/svpablo.html

[22] Aydt R.A.: *The Pablo Self–Defining Data Format*. Technical Report, University of Illinois, Urbana, Illinois 61801, Department of Computer Science, April 1994

[23] Wylie B.J.N., Endo A.: *Design and realization of the Annai integrated parallel environment performance monitor and analizer*, [w:] CSCS–TR–94–07, Centro Svizzero di Calcolo Scientifico (CSCS), August 1994;
http://www.cscs.ch/Official/Software-Tech/CSCS–NEC/PMA.html

[24] Miller B.P., Callaghan M.D., Cargille J.M., Hollingsworth J.K., Irvin R.B., Karavanic K.L., Kunchithapadam K., Newhall, T.: *The Paradyn Parallel Performance Measurement Tool*. IEEE Computer, vol. 28, No. 11, November, 1995, 37–46;
http://www.cs.wisc.edu/~paradyn/

[25] Meira W. Jr.: *Modeling performance for parallel programs*, [w:] Technical Report TR 589, University of Rochester, June 1995;
http://www.cs.rochester.edu/u/leblanc/prediction.html

[26] Wu C.E., Franke H.: *UTE: A unified trace environment for IBM SP systems*, [w:] Technical Report RC 20048(88654), T.J.Watson Research Center, Yourktown Heights, NY, May 1995

[27] Bodin F., Beckman P., Gannon D., Gotwals J., Narayna S., Srinivas S., Winnicka B.: *Sage++: an object–oriented toolkit and class library for building Fortran and C++ restructuring tools,* [w:] *Proc. 2nd Annual Object–Oriented Numerics Conference*, 1994

[28] Mohr B.: *SIMPLE – User's Guide Version 5.3*. Technical Report 3/92, 1–270, Universitaet Erlangen–Nuernberg, IMMD VII, 1992; http://www7.informatik.uni–erlangen.de/tree/IMMD–VII/Research/Groups/MMB/simple/

[29] Wismueller R., Ludwig T.: *The TOOLSET – An Integrated Tool Environment for PVM*, [w:] Liddell H., Colbrook A., Hertzberger B., Sloot P., (eds.), Proc. Int. Conf. High, "*Performance Computing and Networking*", Brussels, Belgium, April 1996, Springer–Verlag Lecture Notes in Computer Science 1067, 1996, 1029–1030;
http://wwwbode.informatik.tu–muenchen.de/Par/tools

[30] Bemmerl T., Bode A., Braum P., Hansen O., Treml T., Wismueller R.: *The Design and Implementation of TOPSYS*. TR TUM–INFO–07–71–440, Technische Universitaet Muenchen, July1991

[31] Ludwig T., Wismueller R., Sunderam V., Bode A.: *OMIS – On-line Monitoring Interface Specification (Version 2.0)*. Technical Report TUM–I9733, SFB–Bericht Nr. 342/22/97 A,Technische Universitaet Muenchen, Munich, Germany, July 1997
http://wwwbode.informatik.tu–muenchen.de/~omis/

[32] Mohr B., Brown D., Malony A.: *TAU: A portable Parallel Program Analysis Environment for pC++*. Proc. of CONPAR 94 – VAPP VI, Linz, Austria, September 1994}, Springer Verlag, 1994;
http://www.cs.uoregon.edu/research/paracomp/tau/

[33] Bubak M., Funika W., Mościński J.: *An overview of performance monitoring and visualizing tools on networks of workstations – looking for generic tools*. Raport wewnętrzny, Instytut Informatyki AGH, Listopad 1994

[34] Reed D.A.: *Experimental Performance Analysis of Parallel Systems; Techniques and Open Problems*, [w:] *Proceedings of the 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation Journal of Parallel Programming*, 16(6), 1994

[35] Hackstadt S.T., Malony A.D.: *Next–Generation Parallel Performance Visualization: A Prototyping Environment for Visualization Development*, Technical Report CIS–TR–93–21, Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, October 1993

[36] Fahringer T.: *Estimating and optimizing performance for parallel programs*, [w:] *IEEE Computer* 28(11) 47–56, November 1995

[37] Wismueller R., Oberhuber M., Krammer J., Hansen O.: *Interactive debugging and performance analysis of massively parallel applications*. Parallel Computing, vol.22, No. 3, March 1996,415–442

[38] Bubak M., Funika W., Mościński J.: *Monitoring of performance of PVM applications on virtual network computer*, [w:] Waśniewski J., Dongarra J., Madsen K., Olesen D., (Eds.), *Applied parallel computing – industrial computation and optimization*. Proc. Third International Workshop, PARA'96, Lyngby, Denmark, August 18–21, 1996, Lecture Notes in Computer Science, 1184, Springer, 1996,147–156,

[39] Bubak M., Funika W., Mościński J.: *Performance Analysis Environment for Parallel Applications on Networked Workstations,* [w:] *Hertzberger B.,Sloot P., (Eds.), Proc. Int. Conf. High Performance Computing and Networking, Vienna, April 28–30, 1997*, Lecture Notes in Computer Science 1225, pp.1002–1005, Springer, 1997

[40] Bubak M., Funika W., Mościński J.: *Evaluation of Parallel Application's Behavior in Message Passing Environment,* [w:] *Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of 4th European PVM/MPI Users' Group Meeting, Cracow, Poland, November 1997*, Lecture Notes in Computer Science 1332, Springer, 1997, 234–241

Recenzent
prof. dr hab. inż. Jacek Mościński