Tomasz WIERCIŃSKI, Marcin RADZIEWICZ, Dariusz BURAK
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE, KATEDRA INŻYNIERII OPROGRAMOWANIA
ul. Żołnierska 49, 71-210 Szczecin

Wykorzystanie kompilacji iteracyjnej do optymalizacji warstwy programowej systemów wbudowanych

Dr inż. Tomasz WIERCIŃSKI

Uzyskał stopień doktora w zakresie informatyki w 2007r. w Zachodniopomorskim Uniwersytecie Technicznym w Szczecinie. Aktualnie jest adiunktem w tym Uniwersytecie. Jego zainteresowania naukowe są skoncentrowane na optymalizacji kompilatorów, językach opisu hardwareu oraz automatycznej syntezie hardwareu (VHDL i SystemC).

e-mail: twiercinski@wi.zut.edu.pl

Dr inż. Marcin RADZIEWICZ

Uzyskał stopień doktora w zakresie informatyki w 2009r. w Zachodniopomorskim Uniwersytecie Technologicznym w Szczecinie. Aktualnie jest adiunktem w tym Uniwersytecie. Jego aktualne zainteresowania badawcze są skoncentrowane na technikach optymalizacji oprogramowania procesorów wielordzeniowych i GPU.

e-mail: mradziewicz@wi.zut.edu.pl

Streszczenie

Artykuł dotyczy wykorzystania kompilacji iteracyjnej do optymalizacji warstwy programowej systemów wbudowanych. W oparciu o autorskie narzędzie WIZUTIC zminejszono czas przetwarzania algorytmu szyfrowania DES. Danymi wejściowymi kompilatora są programy sekwencyjne, wynikami programy zrównoleglone zgodnie ze standardem OpenMP oraz zoptymalizowane pod względem lokalności danych. Parametrem kompilacji iteracyjnej jest rozmiar bloku dla transformacji pętli programowej- tiling.

Słowa kluczowe: systemy wbudowane, kompilacja iteracyjna, programowanie równoległe, algorytm DES.

Exploiting iterative compilation in the software layer of embedded systems optimization

Abstract

Embedded systems are special-purpose computers that perform one or few dedicated tasks. They are mostly part of larger electronic devices, such as communication devices, home appliances, office automation, business equipment, automobiles, etc. Complexity of computers has grown tremendously in recent years, because multi-core processors are in widespread use. Parallelized programs must be run on multi-core processors to use the most of its computing power. Exploiting parallel compilers for automatic parallelization of sequential programs accelerates design processes and reduces costs of the designed systems. In this paper there is described a WIZUTIC iterative compiler developed by the Faculty of Computer Science and Information Technology of the West Pomeranian University of Technology. It uses the source code of PLUTO parallel compiler developed at the Ohio State University by Uday Bondhugula. A simulated annealing algorithm is used for finding optimization passes for the given program features. Parameters that are changed in each iteration are tile sizes of loop transformation tiling. Experimental tests are described and the speed-up results obtained for the DES encryption algorithm are given.

Keywords: embedded systems, iterative compilation, parallel programming, Data Encryption Standard.

Dr inż. Dariusz BURAK

Uzyskał stopień doktora w zakresie informatyki w 2007r. w Zachodniopomorskim Uniwersytecie Technologicznym w Szczecinie. Aktualnie jest adiunktem w tym Uniwersytecie. Jego aktualne zainteresowania naukowe są skoncentrowane na kryptografii i optymalizacji kompilatorów.

e-mail: dburak@wi.zut.edu.pl

1. Wstęp

Systemy wbudowane to komputery specjalnego przeznaczenia wykorzystywane przez większe systemy do kontroli sprzętu, np. urządzeń komunikacyjnych, sprzętu gospodarstwa domowego, samochodów. Układy tego typu są przeznaczone do wykonywania jednego bądź wielu ściśle określonych zadań uruchomianych na danej jednostce sprzętowej [1].

Z uwagi na nieustanny postęp technologiczny w dziedzinie architektur systemów wbudowanych obecnie coraz powszechniej stosowane są jednostki wieloprocesorowe charakteryzujące się większymi mocami obliczeniowymi w porównaniu z układami sekwencyjnymi.

Aby wykorzystać moc obliczeniową tego typu systemów należy dokonać zrównoleglenia programów, które będą przez nie przetwarzane oraz dokonać optymalizacji lokalności danych. W przypadku zastosowania ręcznej metody zrównoleglenia jest to zadanie bardzo pracochłonne oraz podatne na błędy. Jednakże radykalne zmniejszenie czasu oraz kosztów tworzenia oprogramowania równoległego jest możliwe- poprzez zastosowanie kompilatora zrównoleglającego.

W celu utworzenia programu wykorzystującego w maksymalnym stopniu moc obliczeniową jednostek wieloprocesorowych należy zastosować nowoczesną technikę kompilacji-kompilację iteracyjną polegającą na wielokrotnym generowaniu kodu wynikowego w oparciu o zmieniające się parametry kompilacji.

W artykule omówiono proces kompilacji iteracyjnej z wykorzystaniem narzędzia WIZUTIC opracowanego w Katedrze Inżynierii Oprogramowania WI ZUT. W aplikacji wykorzystano kod źródłowy nowoczesnego kompilatora zrównoleglającego PLUTO autorstwa Udaya Bondhugula z Ohio State University. W celu ustalenia takich parametrów kompilacji, dla których generowany program wykonywany jest w najkrótszym czasie wykorzystano powszechnie znaną metodę optymalizacji- symulowane wyżarzanie. Jako parametr kompilacji iteracyjnej wykorzystano rozmiar bloku jednej z najważniejszych metod transformacji pętli programowych- tiling [2].

W przypadku kompilatora WIZUTIC danymi wejściowymi są programy sekwencyjne zapisane w języku ANSI C, natomiast wynikami programy zrównoleglone zgodnie ze standardem przetwarzania równoległego- OpenMP zoptymalizowane w kierunku lokalności danych.

W artykule zamieszczono wyniki badań eksperymentalnych wykonywanych z zastosowaniem algorytmu szyfrowania DES.

2. Kompilator zrównoleglający i optymalizujący lokalność danych- PLUTO

Kompilator zrównoleglający i optymalizujący lokalność danych PLUTO autorstwa Udaya Bondhugula z Ohio State University jest przeznaczony do automatycznej transformacji kodu źródłowego

programu wykonywanego sekwencyjnie zapisanego w języku C do kodu zrównoleglonego zgodnie ze standardem programowania równoległego OpenMP (OpenMP C) w oparciu o model wielościenny. PLUTO posiada możliwość wyzyskania gruboziarnistej równoległości przy jednoczesnej optymalizacji lokalności danych. Najnowsza wersja PLUTO (0.6.0 (BETA)) umożliwia regulację najistotniejszych parametrów dla najważniejszych metod automatycznej transformacji pętli (m.in. rozmiaru bloku (ang. tile size) dla metody tiling, poziomu rozwijania pętli (ang. unroll factor) dla metody unrolling). Istotną wadą PLUTO jest istnienie licznych ograniczeń narzuconych przez jego front-end w postaci narzędzia Clan służącego do ekstrakcji modelu wielościennego, które warunkują akceptację jedynie bardzo małego podzbioru języka C przez PLUTO [3, 4].

3. Kompilacja iteracyjna- narzędzie WIZUTIC

Tradycyjny kompilator optymalizujący ekstrahuje informacje zawarte w kodzie źródłowym dzięki analizie przepływu sterowania oraz analizie przepływu danych, a następnie dokonuje na ich podstawie transformacji kodu źródłowego [5, 6]. W celu zwiększenia wydajności generowania kodu konieczna jest jego restrukturyzacja ze źródła do źródła, ażeby umożliwić np. wektoryzację, zrównoleglenie czy optymalizację lokalności danych, przy czym większość tych transformacji dotyczy pętli. Pomimo tego, że istnieje duża ilość transformacji potencjalnie korzystnych dla zwiększenia efektywności przetwarzania kodu istotną trudność stanowi określenie w ścisły sposób transformacji, które powinny zostać zastosowane w danym przypadku, ich kolejności oraz zakresu ich zastosowania (jest to tzw. problem kolejności faz (ang. *phase-order problem*) kompilatora) [7]. Tradycyjne kompilatory problem ten rozwiązują za pomocą odpowiednio dobranej i z góry określonej sekwencji transformacji utworzonej w oparciu o niewielką ilość danych testowych, ale z uwagi na pewne czynniki, między innymi ich ogromną współzależność oraz znaczne skomplikowanie nowoczesnych procesorów nie zawsze przynosi to zadowalające rezultaty. Podkreślić przy tym należy, że właściwe rozwiązanie powyższego problemu jest niezwykle istotne dla zwiększenia efektywności wykonywania aplikacji ze względu na to, że nawet niewielkie różnice w zastosowanych transformacjach kodu źródłowego przekładają się na znaczące różnice czasu jego przetwarzania.

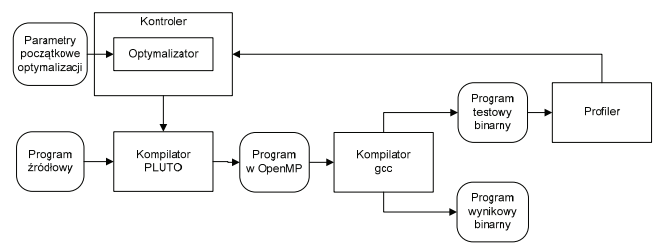
Nowe podejście do powyższego problemu zaproponowali Knijnenburg, Kisuki i O'Boyle [8, 9, 10]. Ich koncepcja nosi nazwę kompilacji iteracyjnej, a celem jej jest zbadanie wieloskładnikowych zbiorów transformacji oraz wybranie na podstawie wyników profilowania kodu przeprowadzonych na maszynie docelowej najlepszego zbioru ze względu na uzyskiwaną szybkość przetwarzania aplikacji oraz rozmiar kodu źródłowego. Podejście takie pozwala uzyskiwać optymalną dla danej architektury sprzętowo- systemowej i dla określonej ilości danych wersję programu poprzez przebadanie wszystkich możliwych wariantów transformacji.

W celu przeprowadzenia reorganizacji tradycyjnego kompilatora optymalizującego i utworzenia na jego podstawie iteracyjnego kompilatora optymalizującego należy dodać dwa dodatkowe komponenty:

- jednostkę kontrolną, która zarządza kolejnością wykonywania poszczególnych faz kompilacji oraz dokonuje analizy uzyskanych informacji;
- jednostkę profilowania kodu, w celu ekstrahowania dynamicznego zachowania kodu wynikowego.

W praktyce możemy mieć do czynienia z bardzo dużym zbiorem możliwych transformacji, co musi mieć przełożenie na znaczną czasochłonność wykonywania wszystkich testowych wersji programu źródłowego i w związku z tym sprawą pierwszorzędnej wagi jest właściwe dobranie zbioru możliwych transformacji kodu źródłowego.

Biorąc pod uwagę wyszczególnione powyżej aspekty zagadnienia opracowano oraz zaimplementowano w oparciu o kod źródłowy kompilatora PLUTO w wersji 0.6.0 (BETA) autorski kompilator WIZUTIC. Poszczególne moduły kompilatora zostały przedstawione na rysunku 1. Narzędzie składa się z modułu kontrolera, który uruchamia kolejne moduły dla każdej iteracji kompilacji. Moduł kontrolera zawiera moduł optymalizacji, który odczytuje parametry początkowe kompilacji oraz ich zakres, a następnie wykonuje kolejne iteracje kompilacji programu wybierając wartości parametrów z dopuszczalnego zakresu przestrzeni optymalizacji z zastosowaniem powszechnie znanej metody optymalizacji- algorytmu symulowanego wyżarzania. Dla każdej iteracji kompilacji uruchamiane są kolejno: kompilator PLUTO, kompilator GNU GCC oraz moduł profilujący kod. Narzędzie PLUTO generuje program zgodnie ze standardem programowania równoległego OpenMP [11] z uwzględnieniem zadanych wartości odpowiednich parametrów kompilacji. Plik zawierający odpowiednie pragmy standardu OpenMP stanowiący daną wyjściową narzędzia PLUTO jest w dalszej kolejności kompilowany z zastosowaniem kompilatora GNU GCC do kodu wykonywalnego dla danej architektury sprzętowo-programowej. Następnie z zastosowaniem jednostki profilującej kod dokonywany jest odczyt metryk uruchomionego programu. Pomiar dotyczy czasu wykonania programu, na który istotny wpływ ma zarówno stopień zrównoleglenia jak i poziom lokalności danych. Otrzymane parametry wykonania programu są następnie analizowane z zastosowaniem modułu optymalizacji. Proces kompilacji iteracyjnej kończy się w momencie osiągnięcia satysfakcjonującej wartości przyspieszenia wykonywania programu.



Rys. 1. Proces kompilacji iteracyjnej kompilatora WIZUTIC
Fig. 1. Iterative compilation process of WIZUTIC compiler

4. Metoda optymalizacji- symulowane wyżarzanie

Do iteracyjnego poszukiwania optymalnych parametrów kompilacji programu najskuteczniejsze są metody dyskretne, niedeterministyczne i znajdujące optimum globalne.

Do tej klasy metod zaliczyć można algorytmy genetyczne, wybrane metody sztucznej inteligencji czy wybrane metody numeryczne.

W narzędziu WIZUTIC zaimplementowano powszechnie znaną metodę optymalizacji- algorytm symulowanego wyżarzania.

Procedura optymalizacyjna rozpoczyna się wygenerowaniem stanu początkowego układu poprzez dobór temperatury T_0 oraz losowych wartości startowych wektora parametrów wejściowych x. W kolejnych iteracjach tworzony jest nowy wektor parametrów x_n. Wartości elementów wektora x_n są losowane metodą najbliższego sąsiedztwa (ang. nearest neighbor method) z przestrzeni iteracji pętli oraz wyliczana jest dla nich wartość funkcji celu $f(x_n)$. Nowe wartości są akceptowane, gdy różnica funkcji celu dla poprzednich i bieżących wartości parametrów jest większa lub równa 0 ($d_{x,xn} = f(x) - f(x_n) >= 0$) oraz gdy losowo wygenerowana liczba k z przedziału [0,1> jest mniejsza od wartości rozkładu Boltzmana dla $d_{x,xn}$ ($\exp(-d_{x,xn} / T)$), gdzie T jest bieżącą temperaturą zmienianą w każdym kroku o ustaloną wartość a ($T = aT_0$).

Szczegółowy opis algorytmu symulowanego wyżarzania zawarty jest w [12].

Wykorzystując powyższy algorytm optymalizujący do znalezienia optymalnego rozmiaru bloku dla transformacji tiling mamy:
1. wartość początkowa (x) to losowo wybrany rozmiar bloku z przestrzeni iteracji,
2. wartości x_n to kolejno losowane rozmiary bloków z przestrzeni iteracji,
3. wartościami funkcji celu $f(x)$ i $f(x_n)$ są czasy wykonania kompilatów dla rozmiarów bloków x i x_n ,
4. początkowa temperatura T to wartość dobrana eksperymentalnie,
5. współczynnik zmiany temperatury a jest wartością dobraną eksperymentalnie.

Zaletami algorytmu symulowanego wyżarzania jest możliwość uzyskania stosunkowo dobrych rezultatów dla niewielkiej ilości próbkowań oraz łatwość implementacji.

5. Algorytm szyfrowania- DES

Problem zwiększenia szybkości przetwarzania algorytmów szyfrowania w realizacjach sprzętowych oraz dla systemów wbudowanych jest obecnie bardzo aktualny, w związku z czym zdecydowano się na wybór reprezentatywnego algorytmu szyfrowania do prowadzonych badań nad wykorzystaniem kompilacji iteracyjnej do optymalizacji warstwy programowej systemów wbudowanych. W tym celu wybrano algorytm Data Encryption standard (DES), który jest w dalszym ciągu popularnym algorytmem szyfrowania, zwłaszcza biorąc po uwagę prowadzone badania nad zwiększeniem efektywności przetwarzania blokowych algorytmów szyfrowania opartych na sieci Feistela (gdyż wiele blokowych algorytmów szyfrowannia opartych na sieci Feistela wywodzi się z algorytmu DES, np. algorytm Triple DES).

Algorytm DES jest przestawieniowo - podstawieniowym algorytmem kryptograficznym wprowadzonym w Stanach Zjednoczonych w 1977 r. przez Narodowe Biuro Normalizacji (National Bureau of Standards).

W algorytmie w trybie szyfrowania DES 64 -bitowy blok wejściowy danych poddany jest permutacji początkowej IP. Po wykonaniu permutacji początkowej blok wejściowy jest dzielony na dwa podbloki: blok lewy - L_0 i blok prawy - R_0. Dalsze operacje wykonywane są oddzielnie dla każdego z tych bloków. Blok L_0 przesuwany jest na lewo, a blok R_0 jest przekształcany zgodnie z funkcją szyfrującą f i zajmuje pozycję po prawej stronie. Operacje powyższe powtarzane są 16 razy. Po zakończeniu ostatniej iteracji łączy się część lewą i prawą, a następnie wykonuje się permutację końcową IP^{-1} (odwrotną do permutacji początkowej IP). Szczegółowy opis algorytmu DES zawarty jest w [13, 14].

6. Badania eksperymentalne

Do przeprowadzenia badań eksperymentalnych z zastosowaniem kompilatora WIZUTIC wykorzystano:
- komputer ośmioprocesorowy zawierający czterordzeniowe procesory Intel XEON E7310 1,6GHz, pamięć cache 4MB, pamięć RAM: 32GB,
- kompilator GNU GCC (wersja 4.3.2, opcja kompilowania -O3) z obsługą standardu OpenMP (wersja 3.0),
- system operacyjny Linux (openSUSE 11.1).

Wejście kompilatora WIZUTIC stanowił:
- kod źródłowy w postaci sekwencyjnej implementacji algorytmu DES szyfrującej oraz deszyfrującej dane w trybie pracy ECB (Kod źródłowy zawarty w [15], przekształcono w taki sposób, ażeby umożliwić szyfrowanie oraz deszyfrowanie dowolnej ilości danych. Ponadto z uwagi na ograniczenia kompilatora PLUTO znacznie uproszczono składnię kodu źródłowego w obszarze przeznaczonym do optymalizacji (zawartym pomiędzy wprowadzonymi dyrektywami kompilatora PLUTO #pragma scop i #pragma endscop), m.in. zastępując zmienne wskaźnikowe odpowiednimi tablicami, usuwając obliczenia występujące w indeksach tablic, zastępując operacje przesunięcia bitowego równoważnymi operacjami mnożenia oraz dzielenia).
- rozmiar danych do zaszyfrowania- 20 megabajtów.

Przy pomocy kompilatora WIZUTIC zoptymalizowana została najbardziej czasochłonna pętla algorytmu odpowiedzialna za szyfrowanie danych w postaci bloków zawarta w funkcji des_enc(). Wykorzystano następujące parametry kompilatora PLUTO: --tile –noprevector umożliwiające optymalizację kodu pod kątem lokalności danych. Nie wykorzystano natomiast parametru: –parallel --tile umożliwiającego zrównoleglenie kodu w połączeniu z optymalizacją pod kątem lokalności danych z uwagi na istniejące ograniczenia kompilatora PLUTO, których nie udało się przezwyciężyć.

Na wyjściu kompilatora WIZUTIC uzyskano zoptymalizowany (użyto w tym celu transformacji pętli programowej zawartej w funkcji de_enc()- metodą tiling z doborem parametrów w oparci o algorytm symulowanego wyżarzania) program implementujący algorytm szyfrowania DES.

Testy wydajnościowe algorytmu DES polegały na pomiarze czasów wykonania algorytmu uzyskanego w wyniku kompilacji przy pomocy:
- kompilatora PLUTO z domyślnymi ustawieniami parametrów kompilacji;
- kompilatora iteracyjnego WIZUTIC (parametry modułu optymalizującego: temperatura początkowa- 200, współczynnik zmiany temperatury- 0,9.

Ze względu na długi czas kompilacji ilość iteracji została ograniczona do 25.

Uzyskane czasy wykonania programów zostały przedstawione w tabeli 1.

Tab. 1. Wyniki badań eksperymentalnych
Tab. 1. Experimental investigation results

Algorytm szyfrowania	Czas przetwarzania [s]	Przyspieszenie pętli zawartej w funkcji des_enc()	Rozmiar bloku [MB]
DES sekwencyjny	6,04	1	20
DES optymalizowany (PLUTO)	1.21	5	20
DES optymalizowany (WIZUTIC)	0.75	**8.05**	20

7. Wnioski

W wyniku zastosowania autorskiego kompilatora zrównoleglającego uzyskano kod uniwersalny algorytmu DES zoptymalizowany pod kątem lokalności danych. Wykorzystano transformację pętli programowych tiling z iteracyjnym doborem istotnych parametrów kompilacji. Uzyskane wyniki mogą być wykorzystane w warstwie programowej systemów wbudowanych, gdyż w tym przypadku pożądane są efektywne algorytmy zoptymalizowane pod kątem lokalności danych.

Przyspieszenie najbardziej czasochłonnej pętli algorytmu DES zawartej w funkcji odpowiedzialnej za szyfrowanie danych w postaci bloków uzyskane w wyniku zastosowania kompilatora WIZUTIC wyniosło 8.05 w stosunku do wersji sekwencyjnej, oraz 1.61 w stosunku do wersji zoptymalizowanej z zastosowaniem kompilatora PLUTO.

Uzyskaną wartość przyspieszenia autorzy planują poprawić poprzez:
- wybór innych algorytmów optymalizacyjnych (w zależności od cech kompilowanego algorytmu, np. wielkości przestrzeni iteracji pętli),
- optymalizację z zastosowaniem innych metod transformacji kodu, np. rozwinięcia pętli (ang. unroll and jam), kontrakcji macierzy (ang. array contraction) oraz iteracyjną zmianę ich parametrów.

8. Literatura

[1] Vahid F., Givargis T.: Embedded System Design. A Unified Hardware/ Software Introduction, John Wiley & Sons, Inc., 2002.

[2] Wolfe M.: High Performance Compilers for Parallel Computing. Addison-Wesley, 1996.

[3] Bondhugula U.: Effective Automatic Parallelization and Locality Optimization using the Polyhedral Model, Ohio State University, 2008.

[4] http://pluto-compiler.sourceforge.net/

[5] Aho A.V., Lam M.S., Sethi R., Ullman J.D.: Compilers: Principles, Techniques and Tools, 2nd Edition. Addison-Wesley, 2006.

[6] Allen R., Kennedy K.: Optimizing compilers for modern architectures: A Dependence-based Approach. Morgan Kaufmann Publishers, Inc., 2001.

[7] Almagor L., Cooper K.D. i inni: Finding effective compilation sequences. In: LCTES'04, Wahsington, DC, USA, 2004.

[8] Knijnenburg P., Kisuki T., O'Boyle M.: Iterative compilation. Embedded Processor Design Challenges: Systems, Architectures, Modelling and Simulation – SAMOS, 2002.

[9] Knijnenburg P., Kisuki T., O'Boyle M.: M. Combined selection of tile sizes and unroll factors using iterative compilation. Journal of Supercomputing, 2003.

[10] Knijnenburg P., Kisuki T., Gallivan K.: Cache Models for Iterative Compilation., Proc. Euro-Par, 2001. Springer Lecture Notes in Computer Science 2150, 2001.

[11] OpenMP Application Program Interface, Version 3.0, May 2008.

[12] Kirkpatrick S., Gelatt C.D., Vecchi M.P.: Optimization by Simulated Annealing, Morgan Kaufmann Publishers Inc., 1987.

[13] FIPS 46-3 Data Encryption Standard (DES), U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technolog, 1999.

[14] Menezes A., van Oorschot P., Vanstone S.: Handbook of Applied Cryptography, CRC Press Online, 1996.

[15] Schneier B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, John Wiley & Sons, 1995.

otrzymano / received: 07.05.2010
przyjęto do druku / accepted: 04.06.2010 *artykuł recenzowany*

INFORMACJE

Prof. dr hab. inż. Janusz Mroczka członkiem korespondentem PAN

Środowisko metrologiczne w Polsce z satysfakcją przyjęło wiadomość o wyborze profesora Janusza Mroczki na członka korespondenta Polskiej Akademii Nauk. Jest to pierwszy od wielu lat reprezentant naszego środowiska w PAN. Ostateczne głosowanie w tej sprawie odbyło się na Zgromadzeniu Ogólnym PAN w Warszawie, w dniu 26.05.2010r.

Profesor Janusz Mroczka jest kierownikiem Katedry Metrologii Elektronicznej i Fotonicznej Politechniki Wrocławskiej. Jego zainteresowania naukowe obejmują metodologię procesu poznawczego, algorytmizację problemu odwrotnego, analizę spektralną i analizę polaryzacji promieniowania rozproszonego w opisie właściwości układów dyspersyjnych, metodologię łączenia danych pomiarowych o różnej przestrzennej rozdzielczości z wykorzystaniem deterministycznych i stochastycznych metod przetwarzania, wykorzystanie reprezentacji czasowo-częstotliwościowych sygnałów w przetwarzaniu danych pomiarowych, opracowanie metody momentów w analizie układów dyspersyjnych.

Jego dorobek naukowy obejmuje 271 pozycji (w tym 41 objętych listą filadelfijską). Jest autorem lub współautorem 4 podręczników i monografii, 11 książek (w tym 4 zagranicznych), 8 patentów. Jego prace znajdują uznanie międzynarodowe (51 cytowań zarejestrowanych w SCI). Wypromował 16 doktorów, opracował 9 recenzji prac doktorskich, 32 recenzje prac habilitacyjnych, 20 recenzji wniosków profesorskich oraz 5 recenzji książek. Był kierownikiem lub wykonawcą 26 grantów, w tym 6 zagranicznych.

Jest członkiem wielu Towarzystw Naukowych, krajowych i zagranicznych. W bieżącej kadencji jest przewodniczącym Komitetu Metrologii i Aparatury Naukowej PAN. Został odznaczony m.in. Krzyżem Kawalerskim Orderu Odrodzenia Polski, Medalem Komisji Edukacji Narodowej, Złotą Odznaką Politechniki Wrocławskiej z Brylantem oraz Medalem im. Prof. Kazimierza Idaszewskiego. Otrzymał Subsydium Profesorskie FNP za rok 2005 pt. "Metrologiczne uwarunkowania fotonicznych metod analizy spektralnej i polaryzacyjnej promieniowania rozproszonego w układach dyspersyjnych". Prowadzi intensywną współpracę naukową z ośrodkami zagranicznymi, takimi jak: Institut Universitaire des Systemes Thermiques Industriels (Mareseille, Francja), Laboratoire d'Energetique des Systemes et Pocedes, Institut National des Sciences Appliquees deRouen (Francja), Departament of Biomedical Engineering, University Boston (USA), Department of Engineering and Product Design, University of Central Lancashire (Preston, Wielka Brytania).

Jest członkiem Rady Programowej miesięcznika Pomiary Automatyka Kontrola.